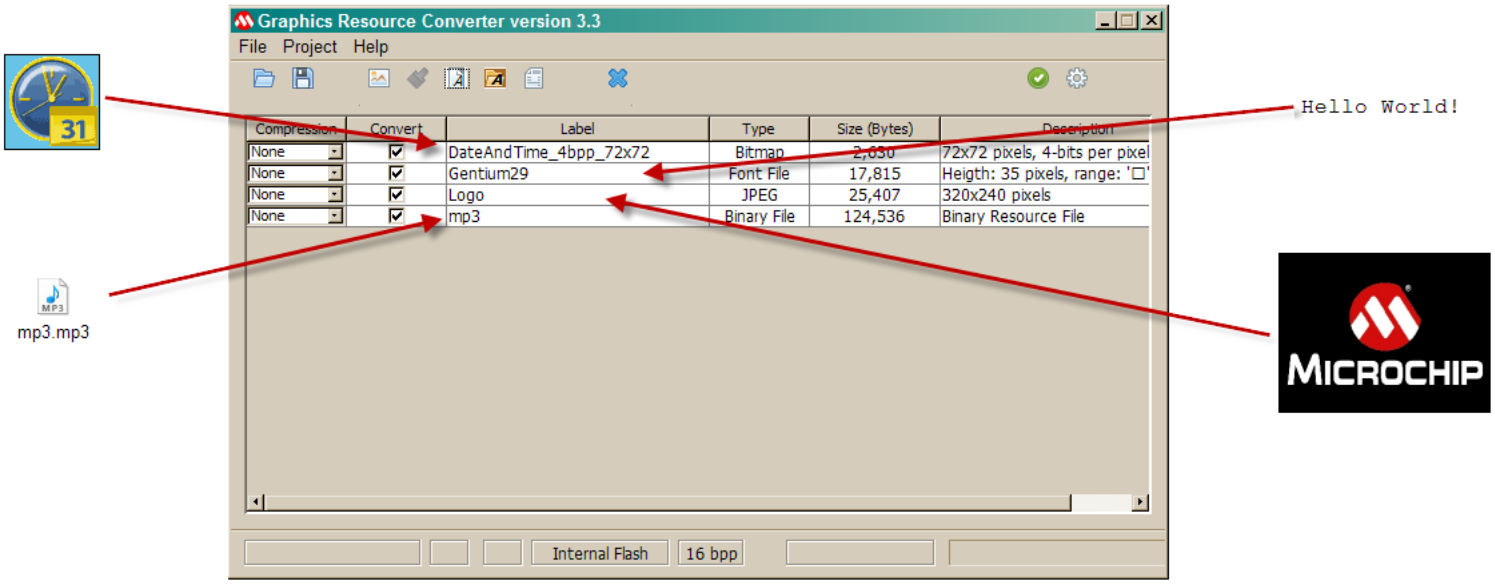
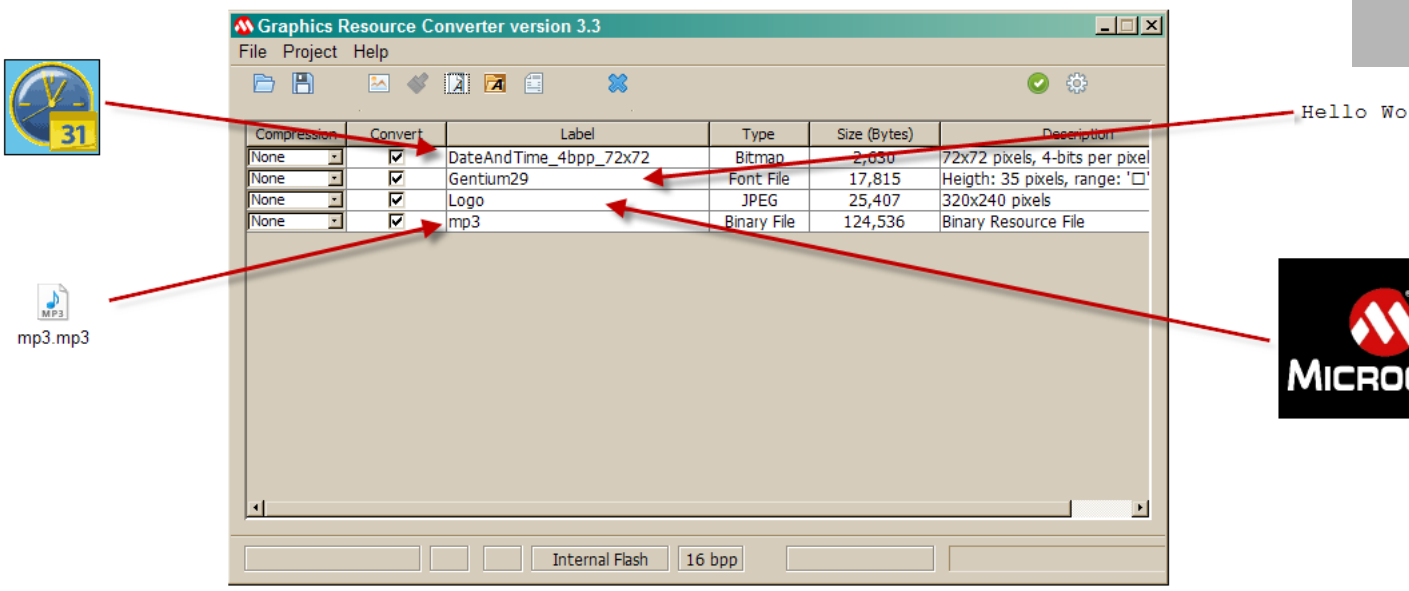


Graphics Resource Converter

Table of Contents

1 Graphics Resource Converter



The Graphics Resource Converter converts images, bitmaps (BMP extension) and JPEG (JPG or JPEG extension), fonts, operating system's installed fonts or True Type fonts directly from files (TTF extension), and binary files into formats to be used with Microchip Graphics Library. Bitmap and fonts are converted to use an optimized encoding for PIC microcontroller usage, while the encoding of JPEG images are maintained. The converter can also be used to create color palettes used by Microchip's graphics module based upon GIMP palettes or bitmaps.

Fonts maybe a copyrighted material so please ensure that you have the rights to use it. You may find free fonts distributed under Open Font License (OFL) agreement. Some of the fonts distributed under OFL may be found here http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&item_id=OFL_fonts.

Importing fonts into application can be performed in two ways. The first method is to identify a range of characters that you want to use and create a font table starting from the first character up to the last character in the range. The second method is to create a character filter file and based on the filter create a reduced character font table. The second method is effective in implementing multi-language applications using only a fraction of the memory required for a full font table implementation. This is especially true for Asian fonts such as Chinese, Japanese and Korean fonts.

Importing images that are not BMP or JPEG requires a step to convert images to a BMP or JPEG format. Multiple image editors that convert other formats to BMP or JPEG format are readily available from software vendors. Microsoft's Paint application is one such image editor. For advanced image editing using an application called GIMP (www.gimp.org) is recommended because it supports capability to do generate optimized color palette for image which can reduce image size.

Importing binary files requires the file to have a .bin extension. Any file to convert in a raw binary format can be renamed to have the .bin extension and loaded into the converter.

Creating a palette can be done in a number of ways, GIMP palette or bitmaps. You can have the converter generate a palette based on the bitmaps that are currently loaded into the resource table. The converter will re-format the bitmaps to use the generated palette table.

2 SW License Agreement

Software License Agreement

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

You may obtain a copy of the License at www.microchip.com/mla_license

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

3 Release Notes

Microchip Graphics Resource Converter

This application is based on the JAVA programming language. To effectively run this program, the computer must have JRE 6 installed.

Please refer to the grc_release_note.txt file under the GRC installed directory.

4 Terms

The following terms and there abbreviations will be used throughout this help file document.

Graphical User Interface (GUI) - A type of user interface that allows user to interact with applications with images rather than text commands.

Command Line Interface or Command Line - A type of user interface that allows for the user or other application to interact with the application with text commands through a command line prompt.

Bits per Pixel (bpp) - The number of bits per pixel, 1, 2, 4, 8, 16 or 24. For example if a picture is 16 bpp, then each color will require two bytes. This value is important when computing the overall memory required by an image.

Microchip Graphics Library - Part of the Microchip Applications Library (MLA). The software allows the users to interface with a number of graphics controllers, both external and internal, to render objects, images and fonts.

Graphics Resource Converter (GRC) - A cross platform JAVA based utility that provides a GUI and Command Line interface for conversion of images, bitmap and JPEG, fonts, binary and palette files into a format which the Microchip Graphics Library can use as part of it's operations.

Resources - An image, font, binary or palette file that can be converted. They are loaded into the GRC for conversion.

Image Resource - Either a Microsoft Bitmap or JPEG file that has been converted for use by the Microchip Graphics Library. The original format of these resources may or may not be modified by the GRC for use with the Microchip Graphics Library.

Font Resource - There are three types of fonts supported by the GRC, Installed, TTF and FNT. Each of the formats are converted into a common structure used by the Microchip Graphics Library. It should be noted that it is not recommended to use installed fonts as they are specific to a system and may not be available on all platforms supported by the GRC. The user may want to copy the TTF file from the installed fonts directory on the operating system and use it.

Binary Resource - This is a resource that contains the raw, unmodified data of a file. Any file may be loaded as a binary, provided that the extension of the file is changed to "bin". It is recommended to first make a copy of the file and then rename the extension.

Palette Resource - Either a bitmap or GIMP palette file. The bitmap file must contain a palette, meaning it is 1, 2, 4 or 8 bits per pixel.

Internal Flash - A type of conversion where the output is a C source and header files. The source file is to be compiled by the firmware application and stored in the PIC's internal flash memory.

External Flash - A type of conversion where the output is an Intel HEX, C source and header files. The information contained in the Intel HEX can be uploaded to an external memory resource. The PIC will use the generated source and header file to retrieve the desired information.

Binary - A type of conversion where the output is a binary, C source and header files. The information of the resources is contained in the binary file. The C source and header files are used to retrieve the desired data.

EDS PMP - A type of conversion where the output is an Intel HEX, C source and header files. The difference between EDS PMP and External Flash conversion, is the address of the chip select is contained in the C source file. This output is to be used with PIC devices with EDS memory space.

Projects - The GRC can save the settings and resources to a XML document.

Graphics Module - The device that uses the converted resources has an internal graphics controller, this option should be selected.

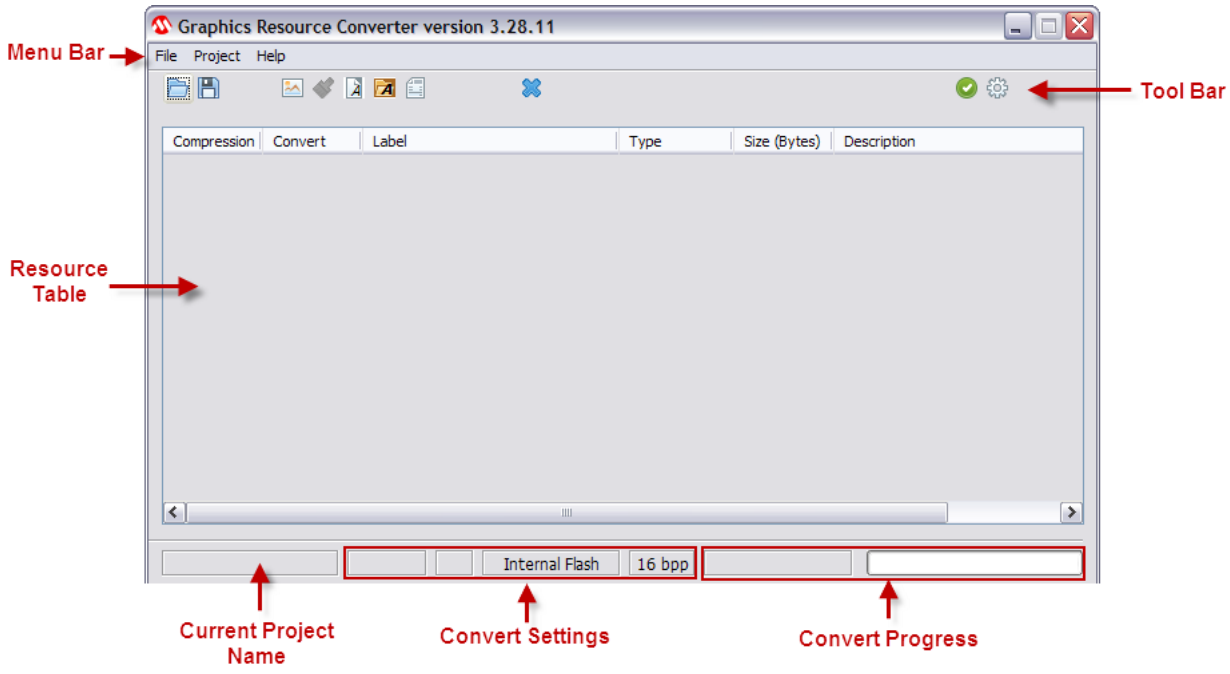
Inflate Processing Unit (IPU) Compression - Uses the Inflate algorithm to decompress data that has been compressed using the Deflate algorithm. Currently the compression method must use static Hoffman tables.

Run-Length Encoding (RLE) Compression - Form of data compression, where data consisting of the same value is stored as a single data value. Commonly used in bitmaps that have palettes with 4 or 8bpp.

Padding - Bitmap Images use padding to make the horizontal lines start on byte boundaries. Padding is used on bitmap images that are less than 8bpp.

5 Graphical User Interface

The main window of the Graphics Resource Converter utility has a table which contains resources which can be converted and several control buttons.



Menu Bar

The Graphics Resource Converter menu bar can be used to add resources like images and font, as well as load and save projects, and there is access to help and support.

Tool Bar Buttons

The tool bar buttons are used to add or remove resources, load and save projects, change convert settings and convert resources.

Resource Table

The resource table displays the current resources of a session.

Current Project Name

This label displays the current project that has been loaded. If there is no project currently loaded, the area will be blank.

Convert Settings

This set of labels displays the convert settings for the project.

Current Converting Resource

When the Graphic Resource Converter is converting resources, the current resource being converted is displayed.

Converting Progress

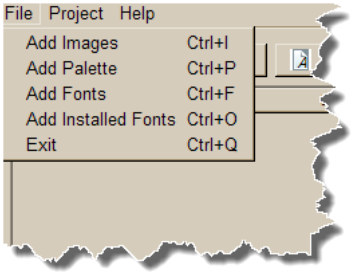
The overall progress of the when converting resources. This is based on the number of files and the relative size of the file to the overall project being converted.

5.1 Menu Bar

Menu Bar

The Graphics Resource Converter menu bar can be used to add resources like images and font, as well as load and save projects, and there is access to help and support.

File



Add Images - Loads an image (BMP, JPG or JPEG) as a resource.

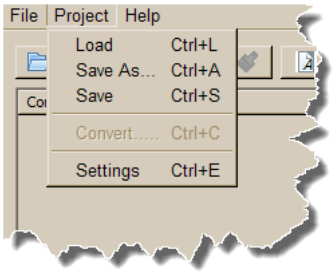
Add Palette - Loads a palette from a bitmap or GIMP file. This feature is only available for PIC microcontrollers that have an internal graphics module.

Add Fonts - Loads a font from a true type font (.ttf) or Windows font file format (.fnt).

Add Installed Fonts - Loads a font from the operating system's list of installed fonts.

Exit - Quits the application

Project



Load - Loads a graphics resource project.

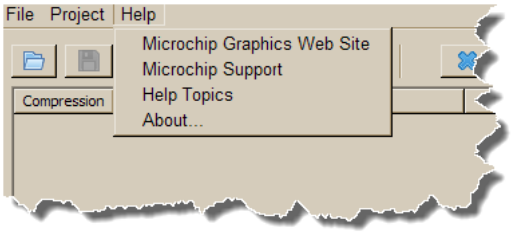
Save - Saves the current project.

Save As - Save the current project under a new project name.

Convert.... - Converts the project

Settings - Opens the settings dialog box.

Help



Microchip Graphics Web Site - Opens a web browser to the Microchip Graphics design center page.

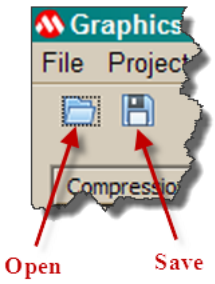
Microchip Support - Opens a web browser to the Microchip Support log in page

Help Topics - Opens the Help documentation for the Graphics Resource Converter.

About.. - Opens the About window.

5.2 Tool Bar Buttons

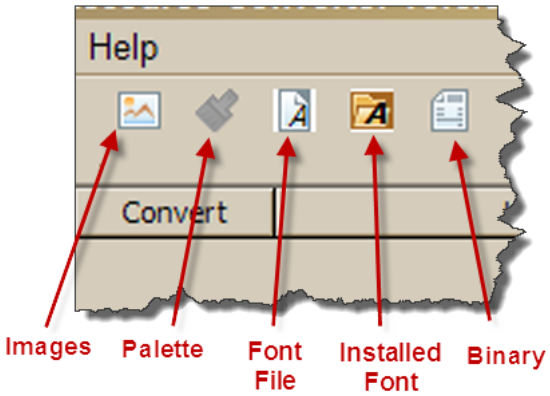
Project



Load - Loads a graphics resource project.

Save - Saves the current project.

Resource



Images - Loads an image (BMP, JPG or JPEG) as a resource.

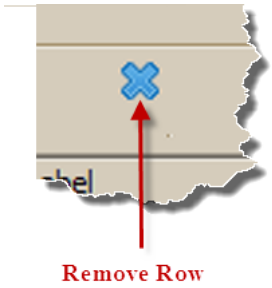
Palette - Loads a palette from a bitmap or GIMP file. This feature is only available for PIC microcontroller that have an internal graphics module.

Font File - Loads a font from a ttf or fnt font file format.

Installed Fonts - Loads a font from the operating system's list of installed fonts.

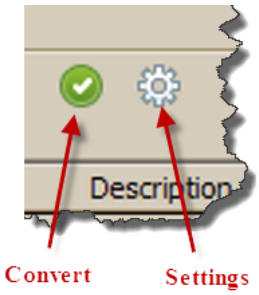
Binary - Loads a raw binary file (.bin) as a resource.

Table Modification



Remove Row - Removes selected row(s) from the resource table. This operation can not be reversed.

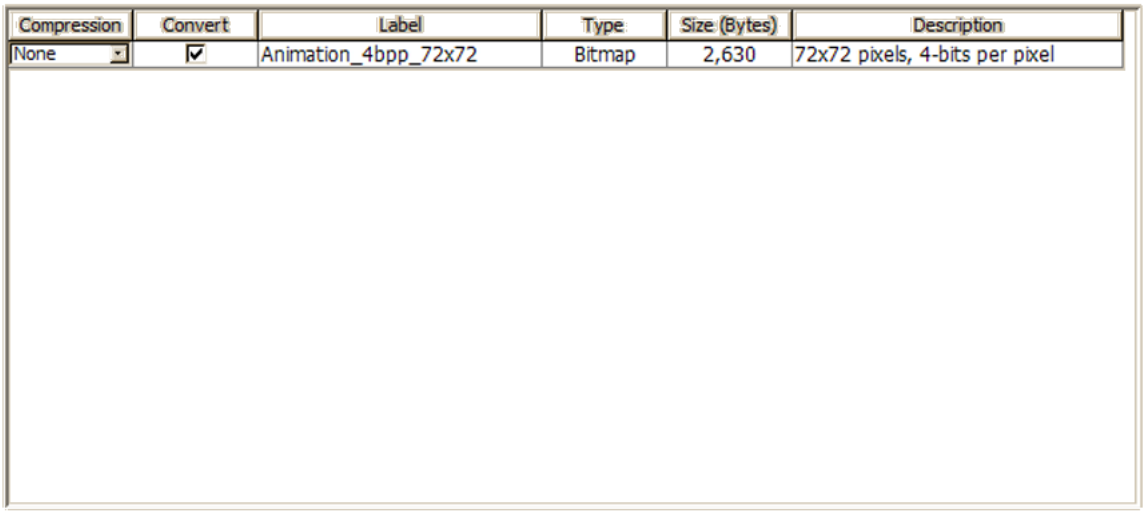
Convert and Settings



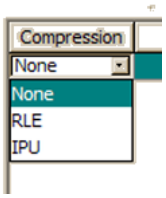
Convert - Converts the selected resources.

Settings - Sets graphics module, converted resource output and the color depth output.

5.3 Resource Table

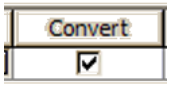


Compression



Indicates if any compression is used to reduce the size of a resource. Not all of the compression options are available for every compiler or module setting. For example, the IPU option is only available with a graphics module.

Convert

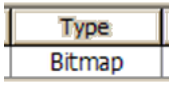


Converting resources, if this box is not checked, the resource will not be converted.

Label

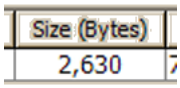
The label that will be given to the resource that the Microchip Graphics Library will reference. For example, the label, *Animation_4bpp_72x72*, will be used by the Microchip Graphics Library.

Type



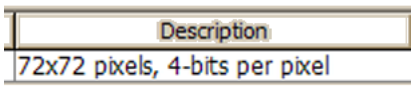
The type of resource to be converted. Valid resource types are Bitmap, JPEG, Palette, Font, and Font File.

Size



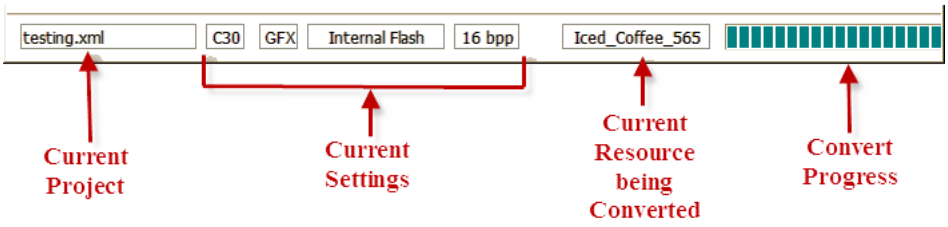
The size, in bytes, of the converted resource. This size is representative of compressed or uncompressed resource.

Description



A description of the resource. For images, the description is the size, in pixels. For palettes, the description is the number of colors. For fonts, the description is the font character range and the height of the font in pixels. The user is responsible for not choosing C syntax names, i.e. char, short, int, or reserved keywords, i.e. for if, else.

5.4 Status



Current Project

Shows the current project of the Graphics Resource Converter.

Current Settings

Shows the current settings of the Graphics Resource Converter, graphics module, what format the converted resources will be saved as and the display bits per pixel.

Current Resource Being Converted

An indicator showing the current resource that is being converted.

Convert Progress

The converting resource progress measured in bytes converted to total bytes to convert.

6 Resources

The following topics describe each of the resources that can be part of a GRC session.

- Image (see page 13)
- Font (see page 16)
- Palette (see page 21)
- Binary (see page 22)

6.1 Image

Image resources consist of Microsoft Bitmap and JPEG images. When a resource image is loaded into the GRC, it will calculate the resource size. The size of the resource may differ from the actual size of the image file. As in the case of a Microsoft Bitmap, the size may be smaller due to converting the image into the Microchip Graphics Library bitmap format.

Microsoft Bitmap Images

The GRC converts the Microsoft Bitmap Image into a format that the Microchip Graphics Library will use to decode the image. The generated data will be less than the image file due to the reduce header size and color depth. The Bitmap Format (see page 13) topic give a detailed overview of the data structure and encoding.

IPU Compression

If the PIC microcontroller has an IPU engine as part of its peripheral set, the bitmap resource can be compressed using the deflate algorithm. The supported deflate algorithm uses static Huffman Tables.

RLE Compression

RLE compression is supported on bitmap resources that have a palette and use 4 or 8bpp.

JPEG Images

The GRC converts all JPEG images as a raw format. The Microchip Graphics Library converts the raw format to display the image.

6.1.1 Bitmap Format

The following tables describe the data structures used by the Microchip Graphics Library to decode the bitmap image.

Bitmap Resource Structure

Block	Name	Bits	Description
Header	Compression	8	Compression of the image, currently set to zero.
	BPP	8	Bits per pixel in the bitmap. Valid values 1, 4, 8, 16, 24.
	Height	16	Image height in pixels
	Width	16	Image width in pixels
Color Table	First color value	16/32	
	...	...	...
	Last color value	16/32	
Raster Data	...	...	...

Bitmap Header Example

```
/**************************************
 * Bitmap header
 **************************************/
    0x00,           // Compresssion
    0x04,           // Color Depth
    0x48, 0x00,     // Height
    0x48, 0x00,     // Width
```

Bitmap Color Table Example: 4bpp using 16bpp format

```
/**********************************
 * Color Palette for the image
 **********************************/
    0x12, 0x33,
    0x2E, 0x2A,
    0x94, 0x4B,
    0x35, 0x5C,
    0xD8, 0x74,
    0x1D, 0xD7,
    0xFA, 0xA5,
    0x4B, 0xA5,
    0x64, 0x73,
    0xA1, 0xEE,
    0x07, 0xF7,
    0x75, 0xFF,
    0x21, 0xC5,
    0x1C, 0x66,
    0xFF, 0xFF,
    0x00, 0x00,
```

Bitmap Color Table Example: 4bpp using 24bpp format

```
/**********************************
 * Color Palette for the image
 **********************************/
    0x96, 0x60, 0x37, 0x00,
    0x70, 0x47, 0x29, 0x00,
    0xA2, 0x73, 0x4B, 0x00,
    0xAF, 0x84, 0x5D, 0x00,
    0xC6, 0x9B, 0x73, 0x00,
    0xED, 0xE0, 0xD4, 0x00,
    0xD6, 0xBD, 0xA3, 0x00,
    0x5E, 0xAA, 0xA6, 0x00,
    0x20, 0x6E, 0x72, 0x00,
    0x0F, 0xD5, 0xED, 0x00,
    0x3D, 0xE0, 0xF4, 0x00,
    0xA8, 0xEF, 0xF8, 0x00,
    0x08, 0xA5, 0xC5, 0x00,
    0xE7, 0xC3, 0x67, 0x00,
    0xFF, 0xFF, 0xFF, 0x00,
    0x00, 0x00, 0x00, 0x00,
```

Color Table Entry Format (16bpp)

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B

Color Table Entry Format (24bpp)

Bits	31	30	29	28	27	26	25	24
Value	0	0	0	0	0	0	0	0

Bits	23	22	21	20	19	18	17	16
Value	R	R	R	R	R	R	R	R

Bits	15	14	13	12	11	10	9	8
Value	G	G	G	G	G	G	G	G

Bits	7	6	5	4	3	2	1	0
Value	B	B	B	B	B	B	B	B

Raster Data Encoding

Pixels are stored left-to-right, top-to-bottom. Color indices are zero based, meaning a pixel color of 0 represents the first color table entry, a pixel color of 255 (if there are that many) represents the 256th entry. For images with more than 256 colors there is **NO** color table.

Raster Data Encoding for 1bit/black & white images

Every byte holds 8 pixels, its LSB representing the leftmost pixel. There are 2 color table entries.

Raster Data Encoding for 4bit/16 color images

Every byte holds 2 pixels, the least significant 4 bits represents the leftmost pixel. There are 16 color table entries.

Raster Data Encoding for 8bit/256 color images

Every byte holds 1 pixel. There are 256 color table entries.

Raster Data encoding for hicolor images

Hicolor images will be down sized to 16bpp if the Graphics Module configuration setting is set to 16bpp, else the color data will be in 24-bit format. For 16bpp data, every 2bytes / 16bit holds 1 pixel, while 24bpp every 4bytes/32bit holds 1 pixel (the high byte is zero for 32bit alignment purposes). The pixels are not color table pointers, rather the actual color to be used. There are no color table entries. Color coded in format in the Color Table Entry Format shown above.

Raster Data Padding

For 1bpp and 4bpp images, padding is placed at the end of a horizontal line to ensure byte aligned data. If non-padding is selected, the horizontal line padding will not be used. For example, if an non-padding image is 1bpp and the width is 14 pixels, the first two bytes of the raster data will have 14 pixel data from the first horizontal line and 2 pixels from the next horizontal line. Likewise, if the image is padded, two bits, zero, will be placed at the end of the 14 pixels of data to ensure that the next horizontal line is byte aligned.

6.2 Font

Font resources consist of System Installed Fonts, True Type Fonts (TTF) and Windows Fonts (FNT). The three different font resources will be converted into a common data structure used by Microchip Graphics Layer. The Font Format topic gives a detailed overview of the data structure used by the Microchip Graphics Layer. To limit the data storage required by the font characters, the GRC allows for a font filter file. This filter file will allow for specific characters or phrases to be converted, thus saving on overall resource requirement. For example the phrase "HELLO" only needs four characters. Using a font filter file will allow for only the four characters to be converted. You can refer to the font filter topic for more information and examples.

Extended Glyph

Unlike ASCII characters, there are Unicode characters which the cursor advancement and the character width are not equal. These character set, mostly Asian, need more information to correctly display a set of characters. Using an extended glyph entry allows for more information than the standard glyph entry. The extended glyph entry contains the glyph width, cursor position, x and y adjustment, as well as the location of the glyph. This extra information allows the graphics library to correct position the next character to the correct position. The Extended Glyph option needs to be selected to generate this information. This feature is available to installed and imported TTF fonts.

Anti-aliasing

Anti-aliasing is supported as a way to reduce the pixelization found in non anti-aliased fonts. The type of fonts supported are installed and TTF fonts. The window font files, FNT, do not support anti-aliasing. The anti-aliasing depth supported is 2bpp. The user may select anti-aliasing on a per font biases. It is recommended not to use anti-aliasing on small fonts under 10 pixels in height.

Font Filter

Name	Description
Font Filter File Format (see page 19)	The font filter file is a text file created in a text editor capable of handling Unicode fonts and saving text files in 16-bit Unicode format. The format of the filter file is shown below:
Font Reference File Output (see page 19)	The font reference file is created to help in the usage of the filtered font table and referencing strings in the application. An example of the output of the font reference file is shown below:
Generating Reduced Font Tables (see page 20)	When using a font filter, the generated font table will only include characters in the strings section of the filter file (see Font Filter File Format (see page 19) for details). To maintain the character ID's of the standard ASCII character table use the special string label "include" and include all the characters in the string from character ID 32 to 127.

6.2.1 Font Format

The following tables describe the data structures used by the Microchip Graphics Library to decode the font data.

Font Data Structure

Font Structure
Font Header

Glyph Entry (First Character)
....
Glyph Entry (Last Character)
Character Glyph (First Character)
....
Character Glyph (Last Character)

Font Header

Name	Bits	Description
Font ID	8	User assignable ID number
Extended Glyph Entry	1	If this bit is set, it indicates that the font uses the extended glyph entry. If this bit is cleared, it indicates that the font uses the standard glyph entry.
Reserved	1	Set to zero
Bits per pixel	2	Bits per pixel (Used for anti-aliasing) For example: 1 = 2^1 or 2 bpp
Orientation	2	Orientation of the character glyph (0, 90, 180, 270 degrees) 0 - Normal 1 - Characters are rotated 270 degrees clockwise 2 - Characters are rotated 180 degrees 3 - Characters are rotated 90 degrees clockwise
Reserved	2	Set to zero
First Character	16	Character code of the first character Example: 32 for ' ' character
Last Character	16	Character code of the last character Example: 126 for '~' character
Height	16	The font characters height in pixels

All characters have the same height. First Character and Last Character define the first and last characters in the font image.

```
/**************************************
 * Font header
 **************************************/
    0x00,           // Font ID
    0x00,           // Font information:  1 bit per pixel, Characters zero degress rotation
    0x20, 0x00,     // First Character
    0x7E, 0x00,     // Last Character
    0x11, 0x00,     // Height
```

Standard Glyph Table Entry

Name	Bits	Description
Width	8	Width of the character glyph and the cursor advance.
Character Glyph Offset: LSB	8	The least significant byte of the glyph location offset.
Character Glyph Offset: MSB	16	The most significant two bytes of the glyph location offset.

The character glyph offset is the index location, in bytes, starting from the font header memory location.

```
/**************************************
 * Font Glyph Table
 **************************************/
    0x07,               // width of the glyph
```

```
    0x84, 0x01, 0x00,       // Character - 32, offset (0x00000184)
```

Extended Glyph Table Entry

Name	Bits	Description
Character Glyph Offset	32	The location offset of the character glyph
Cursor Advance	16	The x-value in pixels which the cursor has to advance for the next glyph
Width	16	The width of the character glyph
X Adjust	16	The X position to adjust
Y Adjust	16	The y position to adjust

The character glyph offset is the index location, in bytes, starting from the font header memory location

```
/*************************************
 * Font Glyph Table
 *************************************/
    0x7C, 0x04, 0x00, 0x00,       // Character - 32, offset (0x0000047C)
    0x07, 0x00,             // cursor advance
    0x00, 0x00,             // width of the glyph
    0x00, 0x00,             // x-position is adjusted
    0x00, 0x00,             // y-position is adjusted
```

Character Glyph

The character is a raster bitmap of the character.

```
/**********************************
 * Character - 49
 **********************************/
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0xC0, 0x01,         //        ***
    0xC0, 0x01,         //        ***
    0xE0, 0x01,         //       ****
    0xF8, 0x01,         //     ******
    0xFC, 0x01,         //    *******
    0xDC, 0x01,         //    *** ***
    0xC4, 0x01,         //    *   ***
    0xC0, 0x01,         //        ***
    0xC0, 0x01,         //        ***
    0xC0, 0x01,         //        ***
    0xC0, 0x01,         //        ***
    0xC0, 0x01,         //        ***
    0xC0, 0x01,         //        ***
    0xC0, 0x01,         //        ***
    0xC0, 0x01,         //        ***
    0xC0, 0x01,         //        ***
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
```

6.2.2 Font Filter

6.2.2.1 Font Filter File Format

The font filter file is a text file created in a text editor capable of handling Unicode fonts and saving text files in 16-bit Unicode format. The format of the filter file is shown below:

```
ButtonStr: ??? // Buttons
CheckBoxStr: ???????? // Checkbox
RadioButtonStr: ?????? //Radio buttons
GroupBoxStr: ???????? //GroupBox
StaticTextStr: ?????? //StaticText
SliderStr: ????? //Slider
ProgressBarStr: ??????? //Progress bar
ListBoxStr: ??????? //List box
EditBoxStr: ?????? //Edit box
MeterStr: ???? //Meter
DialStr: ???? //Dial
PictureStr: ?? //Picture
StaticTextLstStr: ????????
                  ??????
                  ?????
                  ?????????
                  ????????
                  ?????      //Microchip Graphics Library Static Text and Group Box Test.
include:    1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ    //
include:    abcdefghijklmnopqrstuvwxyz              //
include:    "!#$%&'()*+`-.,/:;<=>?@[\]^_"      //dummy string to include the standard ASCII
character set numbers and alphabet.
```

Each line is divided into three sections:

String Label Section	String Section	Comment Section
ButtonStr:	???	// Buttons

1. The string label section - This is the same string label that will be used in the C reference file that will define the character array created for the string it describes. "include" is a special string label that signifies the characters in the string section will be included in the font table but the generated C reference file will not include the string. Note that the character IDs may change so to maintain the character ID of the ASCII characters the whole range of characters from 32 to 127 must be explictly included (as shown in the example above).
2. The string section - The source of the character ID filter to generate the reduced font table.
3. the comment section - This is an optional comment section that users may want to add to the string. The comments are optional but the "//" comment indicator is **required**.

The string label section should be characters using the ASCII codes with identifier names complying with standard C format. This is a requirement since the compiler will not be able to generate code when variable names are not using the standard C formats. The string section will be encoded into an array of 2 byte character ID that the utility will generate. Each line should be terminated by a newline character.

Spaces are counted as characters in the string. Except for the new line character ("/n" or 0x000A), tabs and other control characters are ignored. An example is shown in the "StaticTextLstStr" shown above.

An example of an editor that can be used is the **Word Pad**. Another good editor is the **BabelPad Version 1.9.3**. This is an editor tool available at http://www.babelstone.co.uk/Software/BabelPad.html.

6.2.2.2 Font Reference File Output

The font reference file is created to help in the usage of the filtered font table and referencing strings in the application. An example of the output of the font reference file is shown below:

```
XCHAR ButtonStr[] = {0x00B2, 0x00A6, 0x00BD, 0x0000}; // Buttons
XCHAR CheckBoxStr[] = {0x00A8, 0x009A, 0x00A9, 0x009E, 0x00B2, 0x00A9, 0x009E, 0x00A5,
0x0000}; // Checkbox
XCHAR RadioButtonStr[] = {0x00B8, 0x00A4, 0x009B, 0x00B2, 0x00A6, 0x00BD, 0x0000}; //Radio
```

```
buttons
XCHAR GroupBoxStr[] = {0x009F, 0x00BA, 0x00BE, 0x00B0, 0x00B2, 0x00A9, 0x009E, 0x00A5,
0x0000}; //GroupBox
XCHAR StaticTextStr[] = {0x00EC, 0x00DB, 0x00AA, 0x009D, 0x00A5, 0x00AB, 0x0000};
//StaticText
XCHAR SliderStr[] = {0x00A5, 0x00B8, 0x0099, 0x00A7, 0x00BE, 0x0000}; //Slider
XCHAR ProgressBarStr[] = {0x00B0, 0x00BC, 0x009F, 0x00BB, 0x00A5, 0x00AD, 0x00BE, 0x0000};
//Progress bar
XCHAR ListBoxStr[] = {0x00B9, 0x00A5, 0x00AB, 0x00B2, 0x00A9, 0x009E, 0x00A5, 0x0000};
//List box
XCHAR EditBoxStr[] = {0x00E3, 0x00EB, 0x00B2, 0x00A9, 0x009E, 0x00A5, 0x0000}; //Edit box
XCHAR MeterStr[] = {0x00B5, 0x00BE, 0x00A6, 0x00BE, 0x0000}; //Meter
XCHAR DialStr[] = {0x00A7, 0x0099, 0x00B6, 0x00BA, 0x0000}; //Dial
XCHAR PictureStr[] = {0x00DE, 0x00C7, 0x0000}; //Picture
XCHAR StaticTextLstStr[] = {0x00B3, 0x0099, 0x009E, 0x00BC, 0x00A8, 0x00A9, 0x00B0, 0x0090,
0x000A,
                            0x009F, 0x00B8, 0x00AE, 0x0098, 0x00A9, 0x009E, 0x000A,
                            0x00B8, 0x0099, 0x00AF, 0x00B8, 0x00B9, 0x000A,
                            0x00EC, 0x00DB, 0x00AA, 0x009D, 0x00A5, 0x00AB, 0x0087, 0x0093,
0x0092, 0x000A,
                            0x009F, 0x00BA, 0x00BE, 0x00B0, 0x00B2, 0x00A9, 0x009E, 0x00A5,
0x000A,
                            0x0090, 0x00AA, 0x00A5, 0x00AB, 0x0083, 0x0000};    //Microchip
Graphics Library Static Text and Group Box Test.
```

The character IDs listed in the arrays are not the same as the original IDs. This is due to the fact that the utility will be generating a font table with contiguous and no unused character ID. Unused character IDs in the original font table are removed resulting in memory saving.

6.2.2.3 Generating Reduced Font Tables

When using a font filter, the generated font table will only include characters in the strings section of the filter file (see Font Filter File Format (see page 19) for details). To maintain the character ID's of the standard ASCII character table use the special string label "include" and include all the characters in the string from character ID 32 to 127.

```
include:    " !#$%&'()*+`-.,/:;<=>?@[\]^_{|}~"    //
include:    1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ  //
include:    abcdefghijklmnopqrstuvwxyz // the standard ASCII character set from ID's 32 -
127
```

This will generate the font table with the ASCII characters with ID from 32 to 127. Doing this enables the user to refer to each character in the application code in a normal fashion as shown in the example code below:

```
static char StringName[] = "Hello World";
```

However, if the characters included are not the complete set of the 32 to 127 character IDs, the generated font table may change the character ID's of some or all the characters. For example:

```
include:    1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ    //
include:    abcdefghijklmnopqrstuvwxyz // not the complete standard ASCII character set
from ID's 32 - 127
```

The generated font table will assign a character ID of 33 to '0' (character zero) and not 48 as seen from the ASCII table. The reason is that the range of characters from '!' to '/' was not included. The utility will remove the unused characters and move the next character to the location of the first removed character. The scheme will be performed on all characters thus the generated font table will have a completely new character IDs. When this happens, the code above will not work since the compiler assumes that the string "Hello World" will use the standard character IDs of all the characters. The code must be modified to this form:

```
XCHAR HelloStr[] = {0x0032,0x0049,0x0050,0x0050,0x0053,0x0020,
                    0x0041,0x0053,0x0056,0x0050,0x0048, 0x0000}; //"Hello World";
```

If no other characters in the ASCII set will be used other than the characters that comprises the "Hello World" string it will be best to use a string label to define the string "Hello World" in the font filter file.

```
HelloWorldStr:    Hello World      // generate only these characters in the font table
```

The generated reference file (see page 19) will contain this declaration:

```
XCHAR HelloStr[] = {0x002B,0x0034,0x0037,0x0037,0x0038,0x0020,
                    0x0030,0x0038,0x003A,0x0037,0x0033,0x0000}; // using reduced font table
```

This method frees the user from manually calculating the converted character ID's.

Notice the 0x0020 is the space character. This is the only character that will maintain the character ID since by default the utility always start from the space character. Control characters are omitted from the generated table.

6.3 Palette

Palette Resource consist of Microsoft Bitmap or GIMP palette files. If the palette is derived from a Microsoft Bitmap, the bitmap must contain a palette. The Palette Format topic give a detailed overview of the structure used by the Microchip Graphics Library.

6.3.1 Palette Format

The following tables describe the data structures used by the Microchip Graphics Library to decode the palette data.

Palette Header

Name	Bits	Description
Resource Description	16	See Resource Description Table (see page 52)
ID	16	User assignable ID
Entry Number	16	Number of palette entries, 2, 4, 16, 256
Table Address	Pointer Length (16/24/32)	Pointer to the location of the palette entry table

Palette Entry

The palette entry is a union and can be used three ways

Name	Bits	Description
Value	16	16-bit RGB color value

Color	Bits	Description
Red	5	Red color component of the RGB color
Green	6	Green color component of the RGB color
Blue	5	Blue color component of the RGB color

NOTE: In 16 RGB color, six bits are used for the green component due to the human eye being more sensitive to that color component.

Monochrome	Bits	Description
Luma	4	The monochrome luma value

6.4 Binary

Binary Resource consist of a files raw data. Any file can be a binary resource provided the extension of the file is BIN. It is recommended that a copy of the binary file is made and that copy's extension changed to BIN for use as a Binary Resource.

7 Using the GUI

7.1 Adding Resources

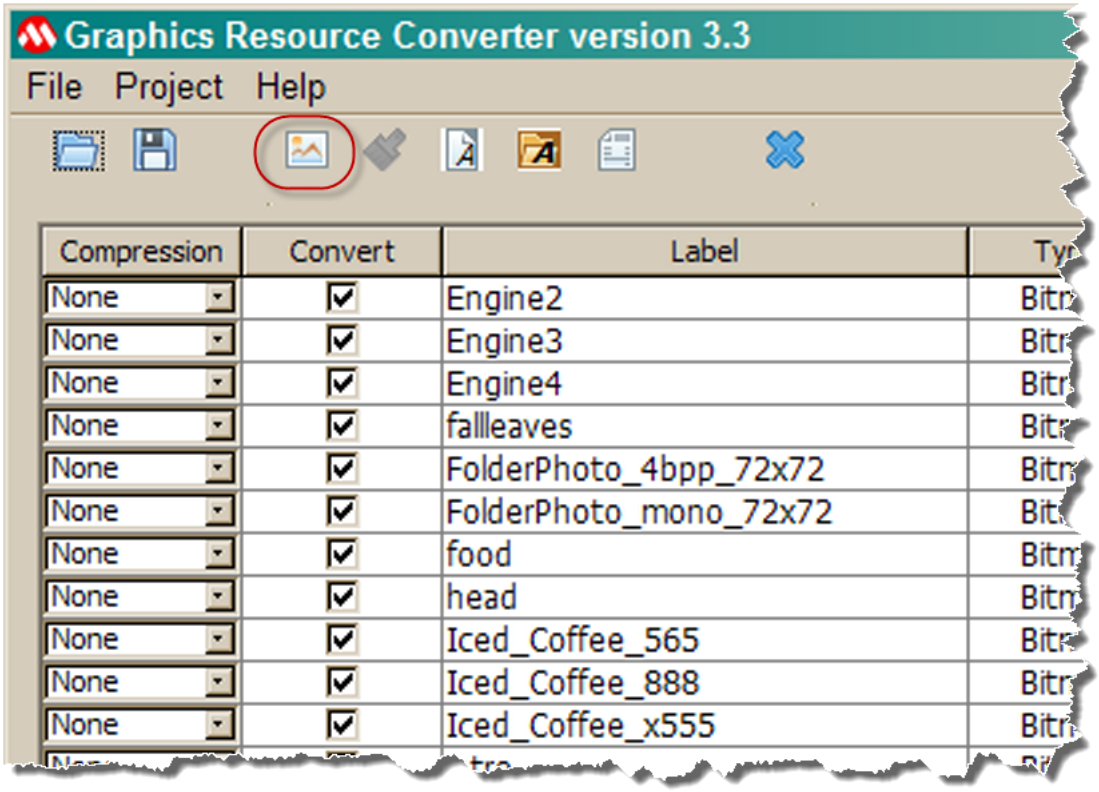
The following topics describe, in detail, how to add a specific resource to the GRC.

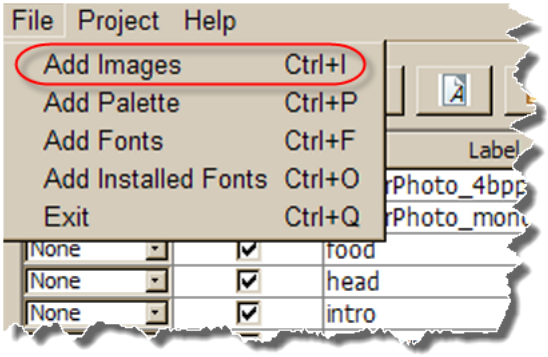
- Add an Image Resource (see page 23)
- Add a Font Resource
  - Installed Font (see page 25)
  - Font File (TTF or FNT) (see page 29)
- Add a Palette Resource (see page 31)
- Add a Binary Resource

7.1.1 Images

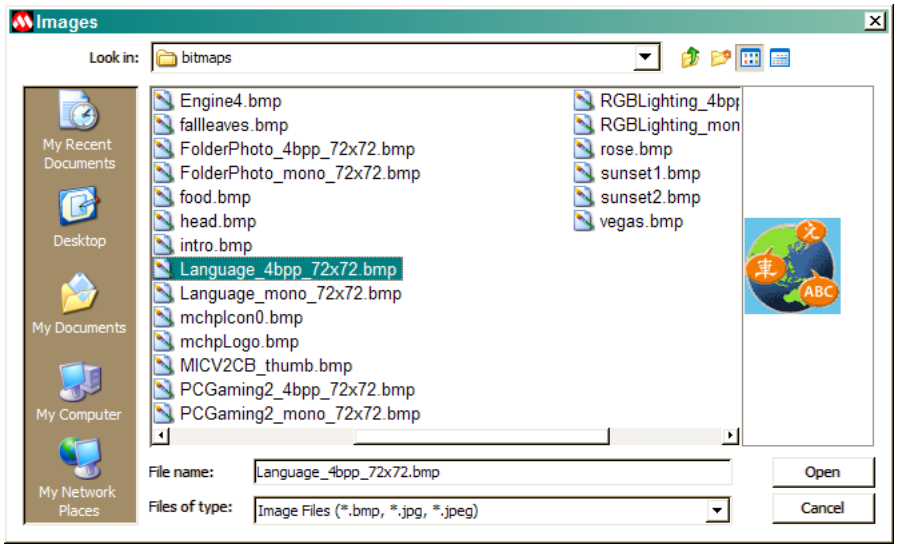
To add an image (bitmap or JPEG format) for conversion the following steps should be done:

1. Press **Add Images** button or File menu item.

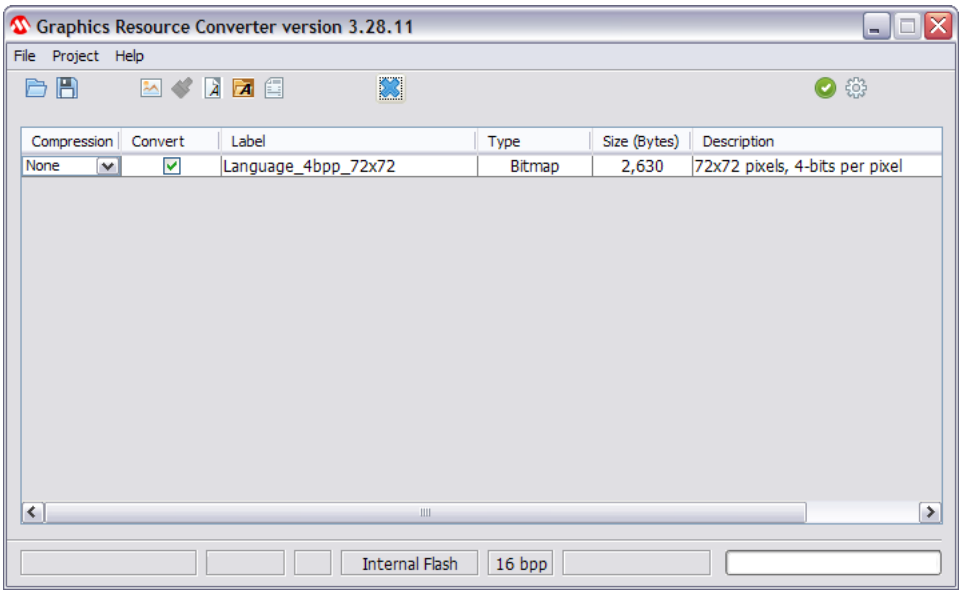




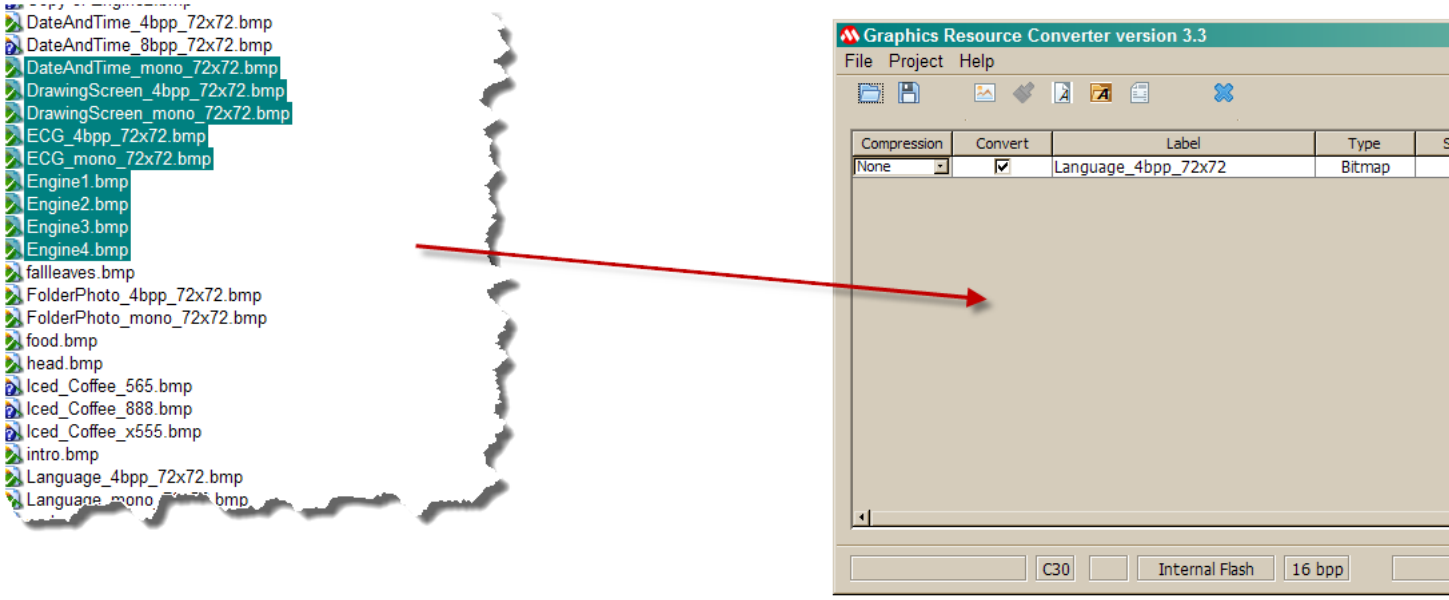
2. A file dialog box will appear. The user can select a single or multiple files to import to the Graphics Resource Convert. When done, select the **Open** button.



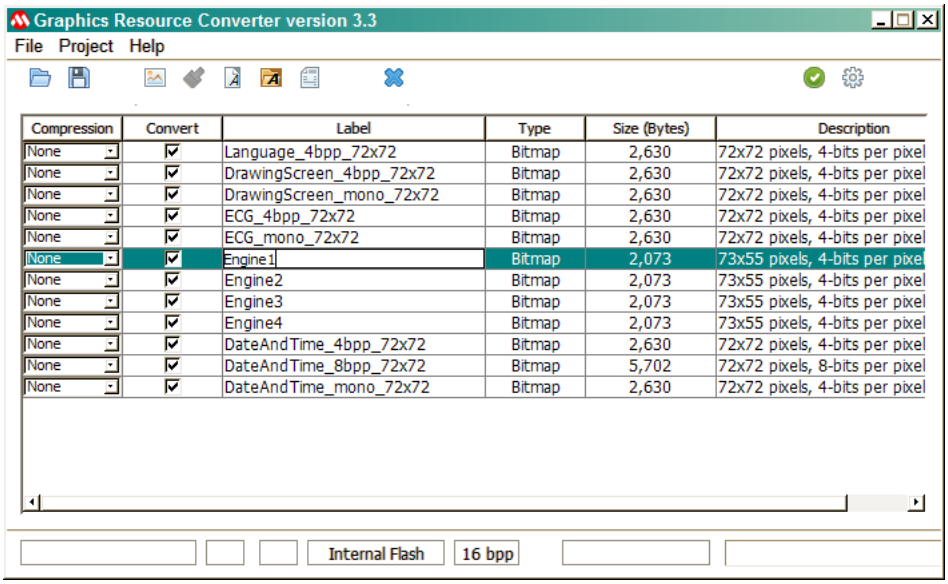
3. If selected file will be imported successfully the information about it (label, type, size and description) will be added in the list box of the main window. Label is generated from the file name, type defines that imported object is using a bitmap or a JPEG format, size of data is shown in bytes, and description contains basic information. Generated label must be used for the reference to this image in the application.



4. Graphics Resource Converter also supports Drag-and-Drop for adding images. Simply select the images files and drag them to the resource table.



4. To change the label double click on it and modify the label. The first label letter cannot be a number.



7.1.2 Installed Fonts

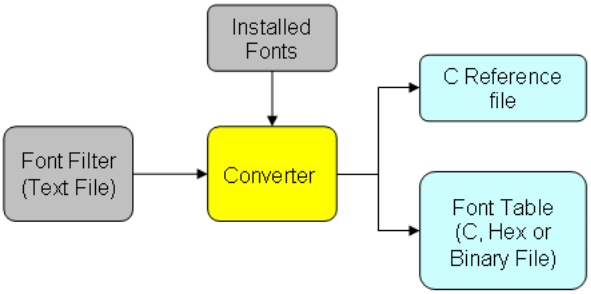
Importing Fonts using a Range of Characters

This option is normally done in languages with character sets of 255 characters or less. A range of characters is defined from one character index to the another character index. An example would be the ASCII character set where the English

characters are defined from character index 32 or 0x20 hex (space character) to character index 126 or 0x7E hex(~ character). The font table generated for the range will contain all the characters from 0x20 to 0x7E inclusive. User's who want to use some other fonts with UNICODE character ranges can still be handled by this utility. The font table generated will still contain the characters with the defined range. Character index that has no defined characters glyph will be either a blank space or assigned to a default character. However, this approach is wasteful in terms of memory and it is recommended to use a font filter.

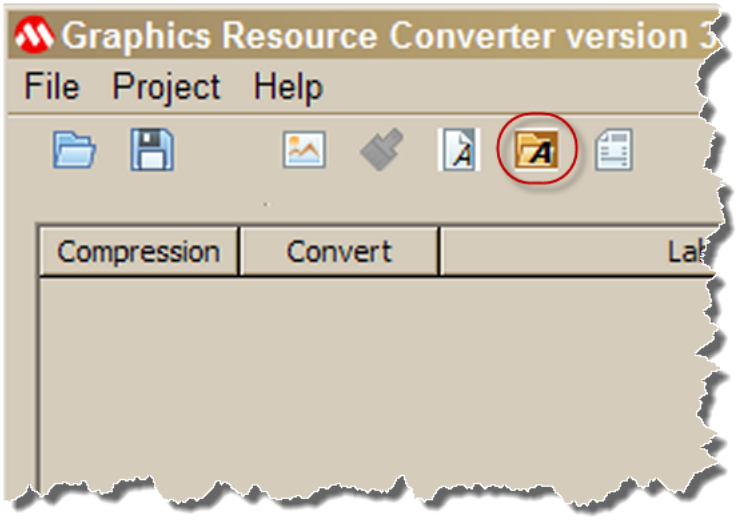
Importing Fonts using Character ID Filter

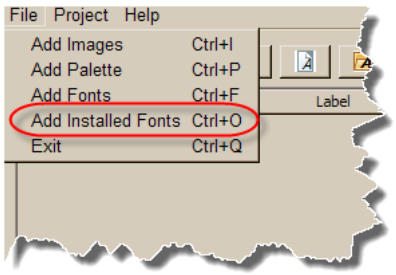
Another way to overcome the problem of memory requirements for fonts with thousands of character indexes is filtering the font table and recreating a reduced character set table which contains only the pre-defined characters. This approach will need a filter text file (described in Font Filter File Format (see page 19) section). The generated font table will contain all the character glyphs of the characters defined in the filter file. Except for the first character (character with the lowest character index) the character indexes of all the glyphs are changed. The utility will also generate a C source reference file to be used in the application to easily refer to the new font table. The figure below shows the general components in generating a reduced font table with its outputs.



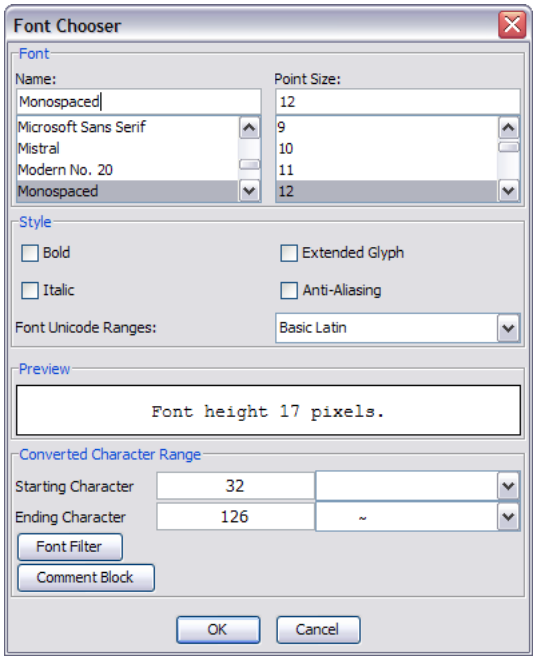
To add an installed font for conversion the following steps should be done:

1. Press **Add Installed Fonts** button or the File menu item. "Font Chooser" dialog will appear.

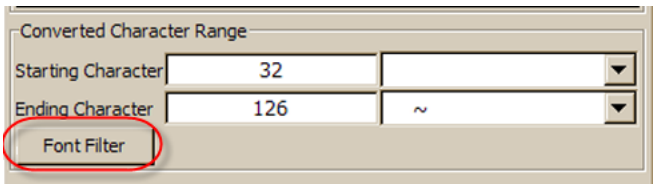


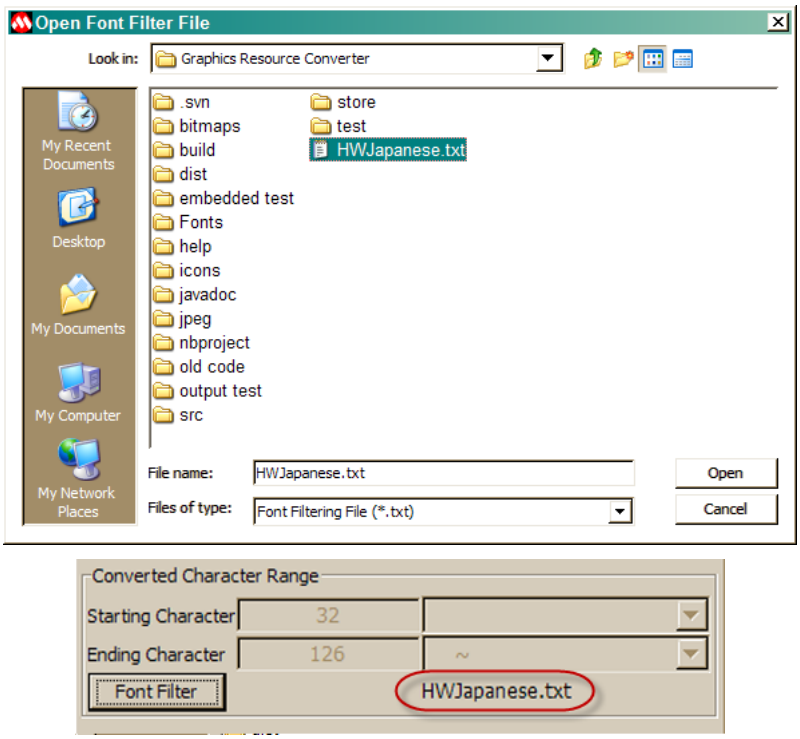


2. In the dialog select the installed fonts that will be used along with the "Font Style", "Font Size", "Unicode Range", "Starting Character", "Ending Character" and "Font Filter" button. An example of the selected font can be previewed with the height of the selected font in pixels.

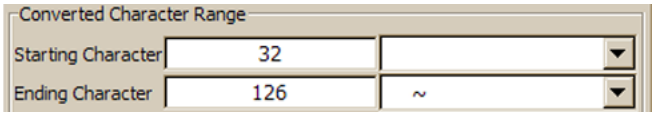


3. A font filter can be selected by pressing the "Font Filter" button. When selecting the "Font Filter" button, a file dialog will appear. This gives the user the opportunity to assign a font character filter to the selected font. This filter will be used to extract character ID's that will comprise the font table.

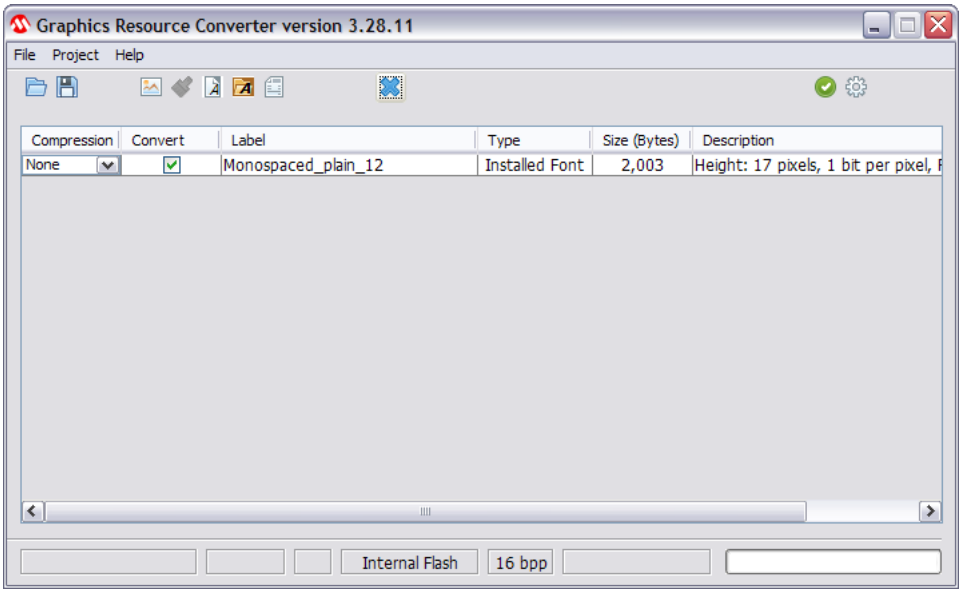




4. If filtering is not used, it is assumed by the utility that the user wants to use a range of characters instead. The user can choose a range of the characters that will be used in the font table. Starting Character sets the character index of the first character in the table. Ending Character sets the character index of the last character in the table.



5. After selecting the font and all of it's attributes, press OK and it will appear in the resource table.



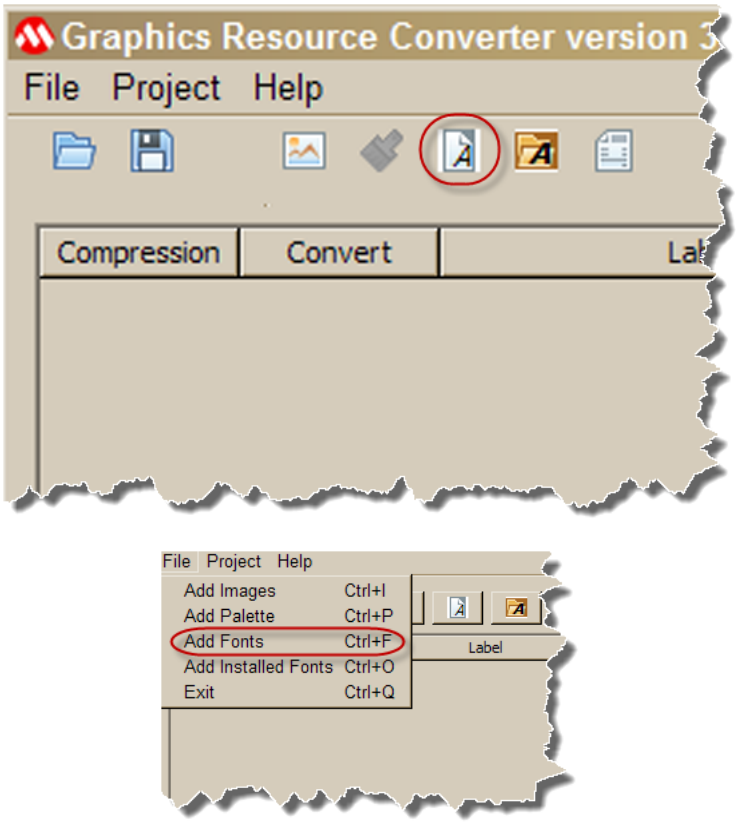
7.1.3 Font Files

Importing Fonts from Files

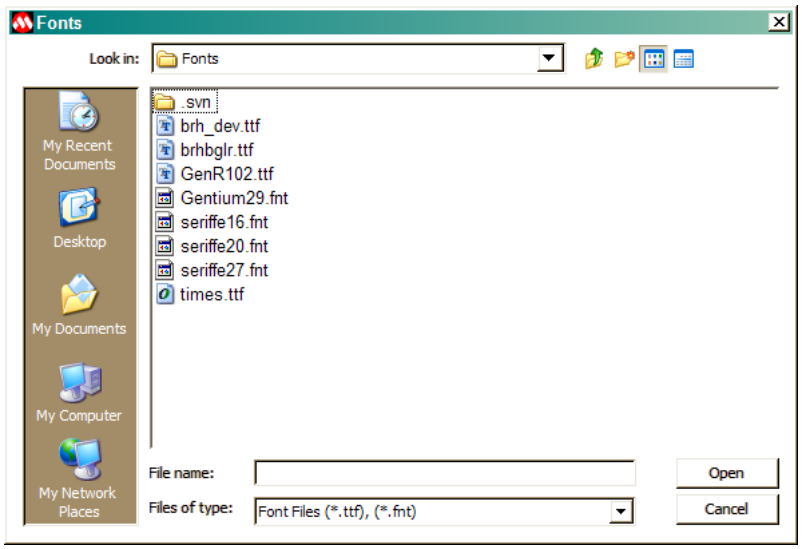
Aside from the installed fonts, the user can also import a font table from a raster font file (*.fnt) or from a true type font file (*.ttf). Raster fonts can only import fonts with character sets ranging from 0-255. True type fonts on the other hand can be imported with the same options as installed fonts. They can be imported using a range of character ID or using a font filter file.

To add fonts using font files as inputs the following steps should be done:

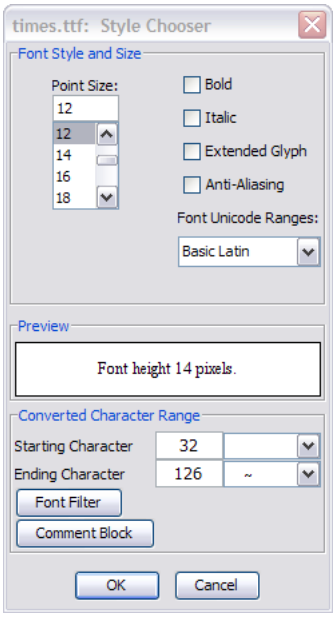
1. Press **Add Fonts** button or File menu item. "Add font" dialog will appear.



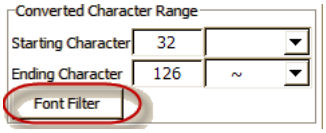
2. In the dialog select the file type extension to select importing raster font (*.fnt) or true type font (*.ttf). Browse and select the file and press **Open**.
   - Go to step 6 to generate raster fonts.
   - Go to step 3 to generate true type fonts.

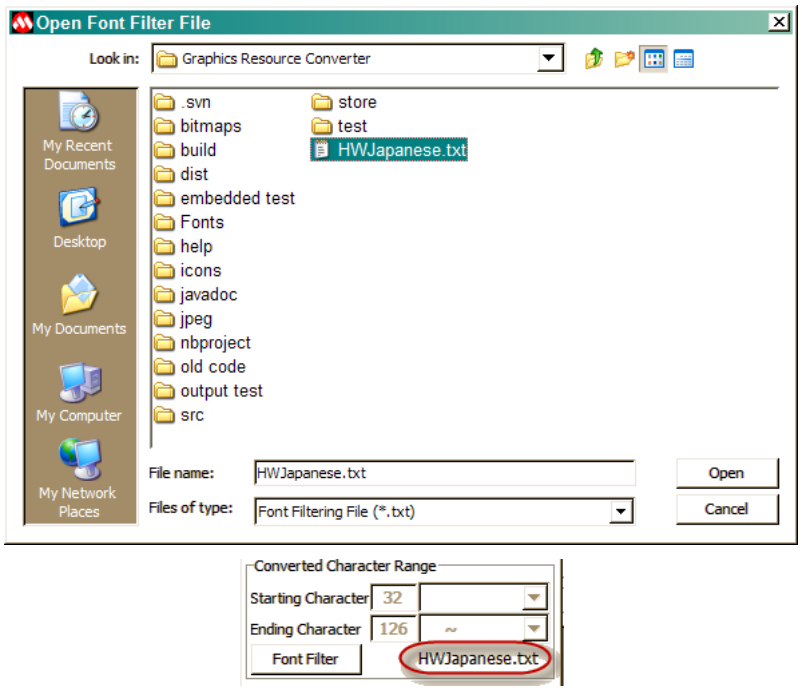


3. "Font Style Chooser" dialog will open. This dialog will show parameters for the selected true type font that can be set. Most importantly it shows the character sets or UNICODE ranges that the font supports.

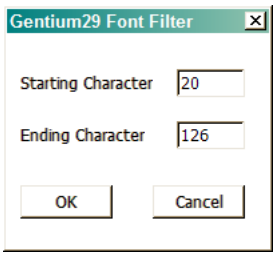


3. A font filter can be selected by pressing the "Font Filter" button. When selecting the "Font Filter" button, a file dialog will appear. This gives the user the opportunity to assign a font character filter to the selected font. This filter will be used to extract character ID's that will comprise the font table.





5. Press **OK** to generate the font table.
6. When raster fonts are selected instead of true type fonts, instead of the "Font Style Chooser" dialog box "Font Filter" dialog box will appear.



7. Set the desired character range and press **OK** to generate the font table.

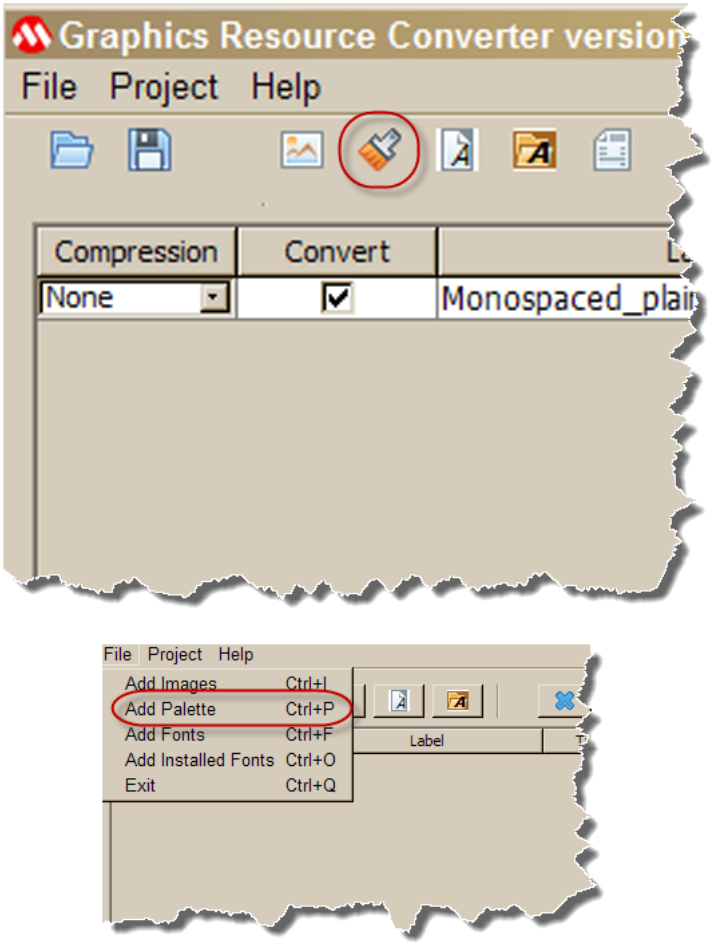
7.1.4 Palettes

Importing Palettes

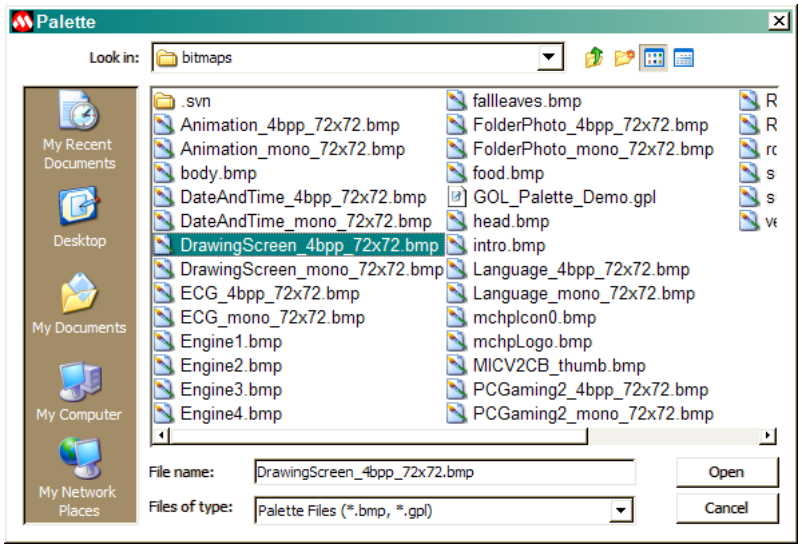
The tool can extract the color table of a bitmap file (bitmaps with 24-bit colors do not have color tables, the tool will not process these files) or load a GIMP palette file (*.gpl). To add a palette to the resource converter, the target must have an internal graphics module.

To load the color table of a bitmap file the following steps should be done:

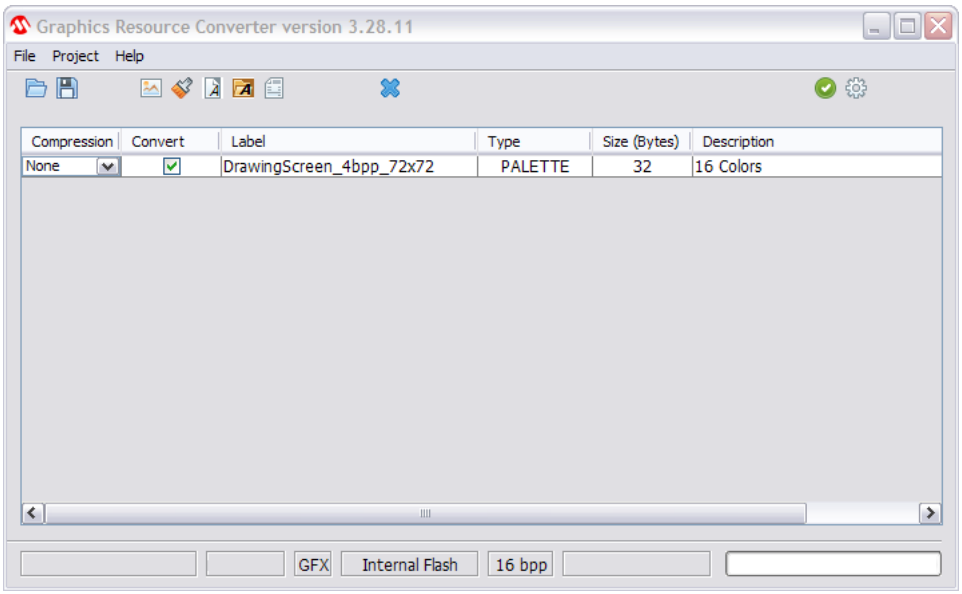
1. Press **Add Palette** button or File menu item. "Open file" dialog will appear.



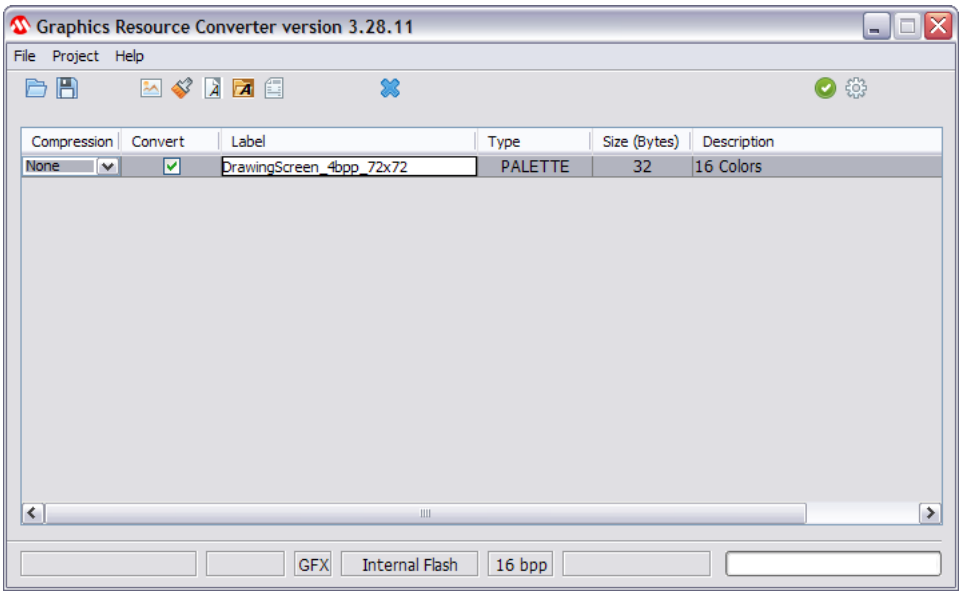
2. A file dialog will appear and by default, filters the Bitmap File (*.bmp filter) or the GIMP Palette file (*.gpl filter).Browse for the image file to be added and press **Open** button.



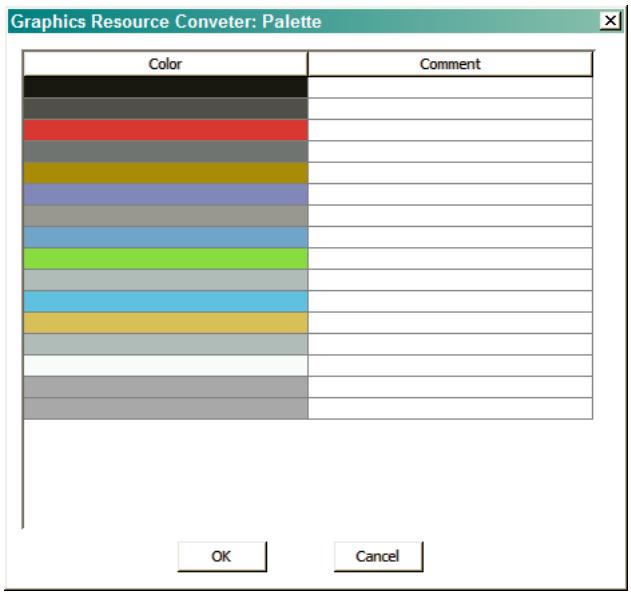
3. If selected file will be imported successfully the information about it (label, type, size and description) will be added in the list box of the main window. Label is generated from the file name, type defines that imported object is a palette, size of data is shown in bytes, and description contains the number of color entries in the palette. Generated label must be used for the reference to this image in the application.



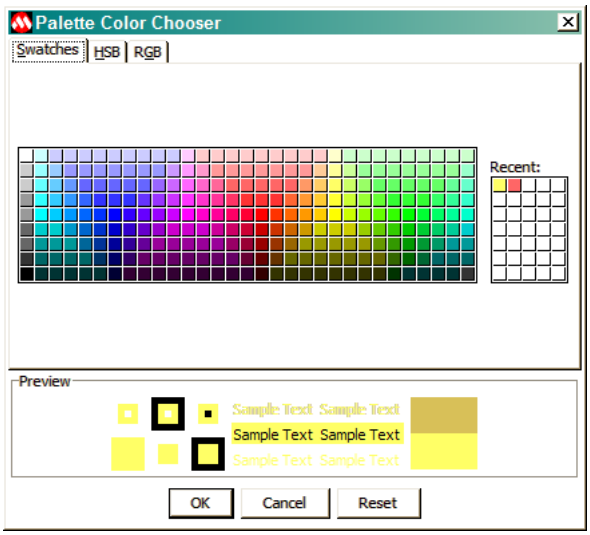
4. To change the label double click on it. Modify the label and press **OK** when done. The first label letter cannot be a number.



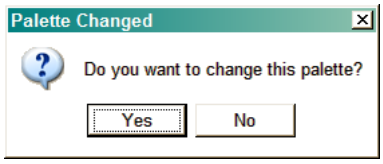
5. By double clicking on the palette entry a dialog box will appear to allow the user to edit the color table.



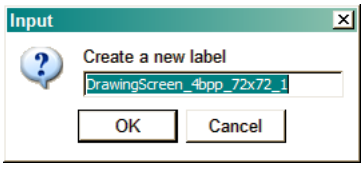
6. Double click on the color entry in the table to edit it. A color chooser will appear. Select the new color and select **OK**.

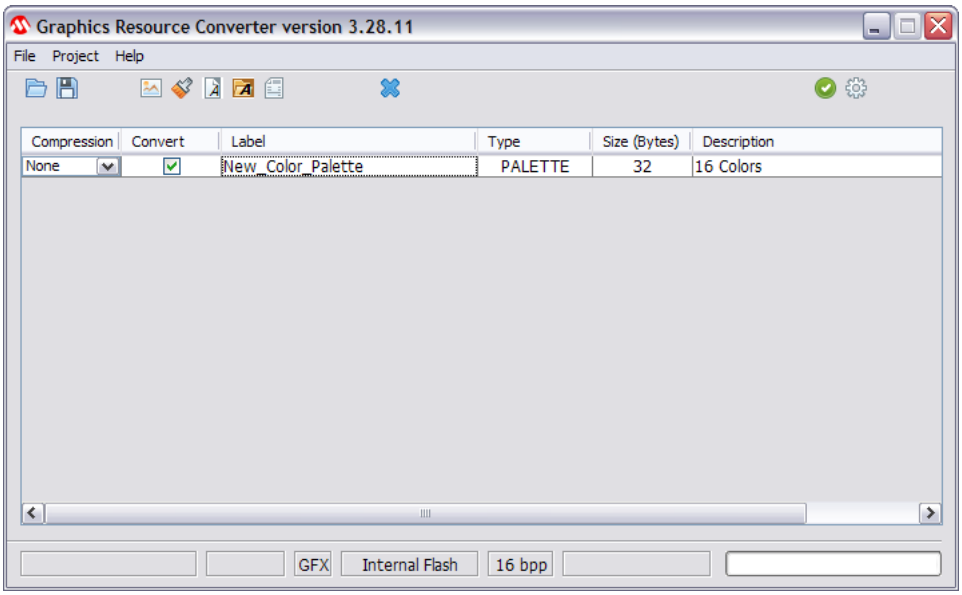


7. After done making edits, select **OK.** If you want to change the palette, select **YES**.



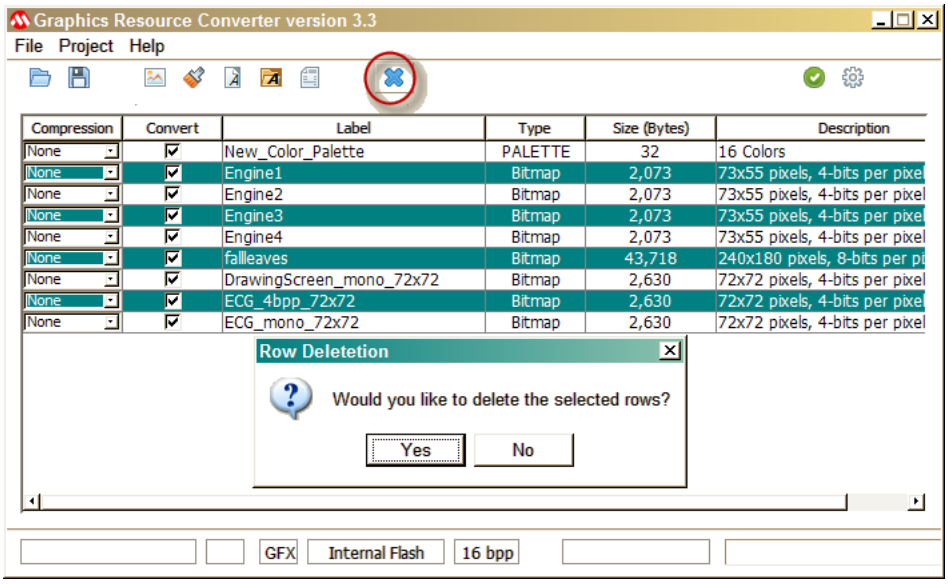
8. An input window will appear to create a new label.

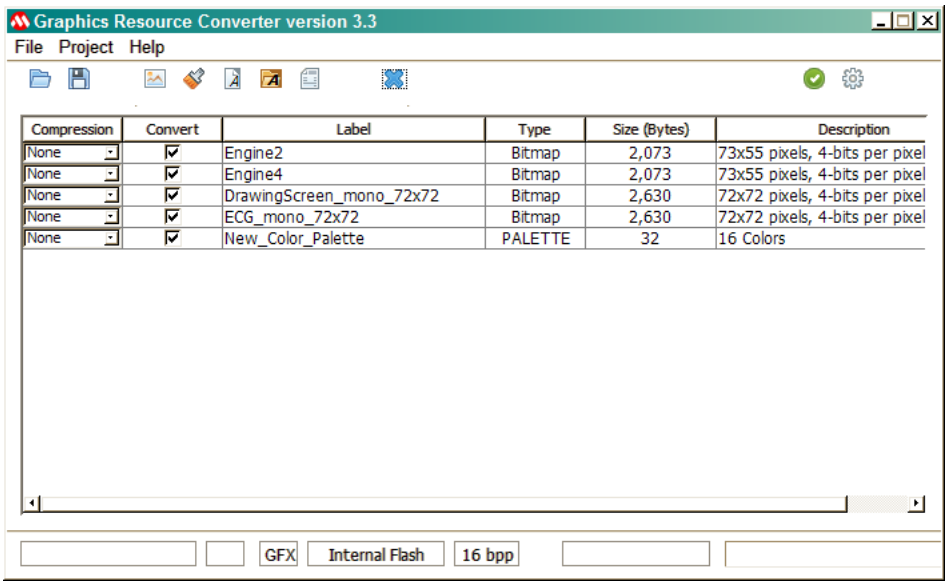




7.2 Removing a Resource

To remove some item from conversion list select the item or items and press **Remove** button. This action can not be undone.





7.3 Projects

The GRC allows for user to load and save projects. The projects contain all resource and settings information.

- Loading a Project (see page 36)
- Saving a Project (see page 38)

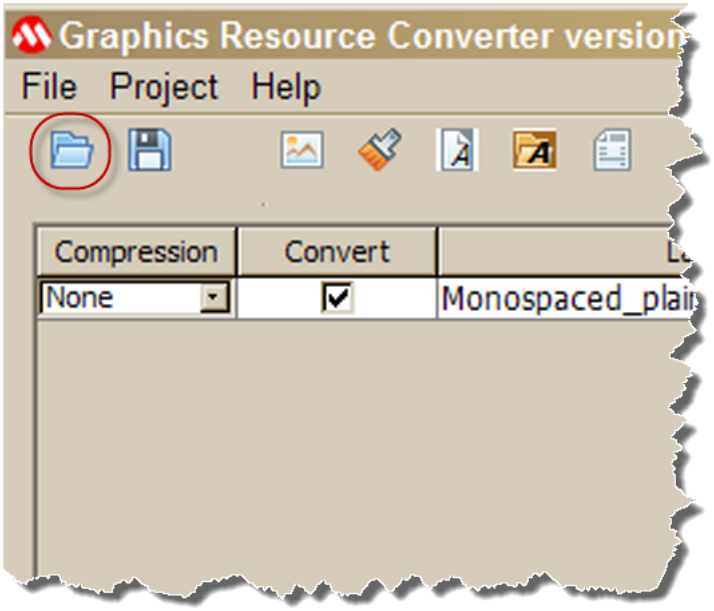
7.3.1 Loading

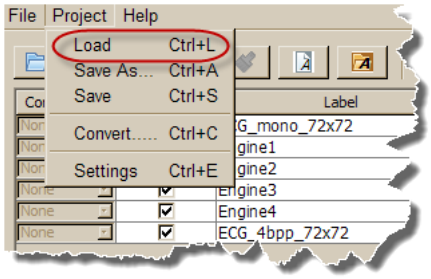
The utility allows users to load previous projects.

Loading a Project

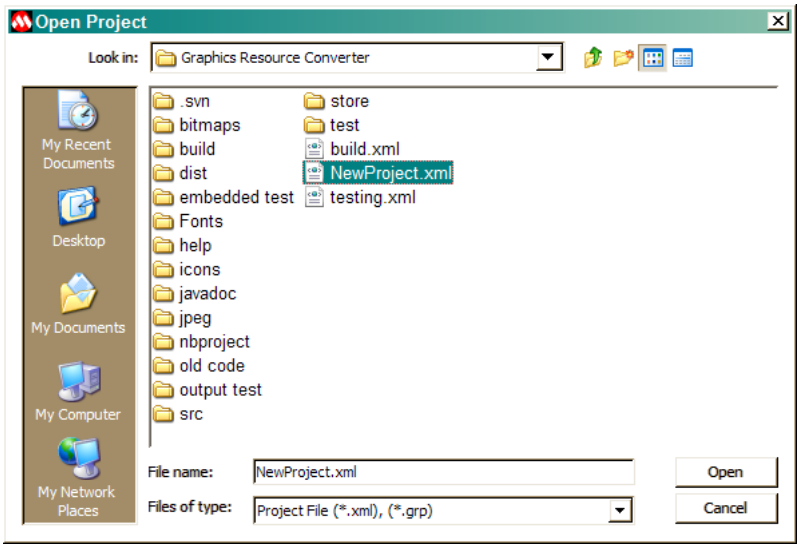
To load a project the following steps should be done:

1. Press **Load** button or the Project menu item.

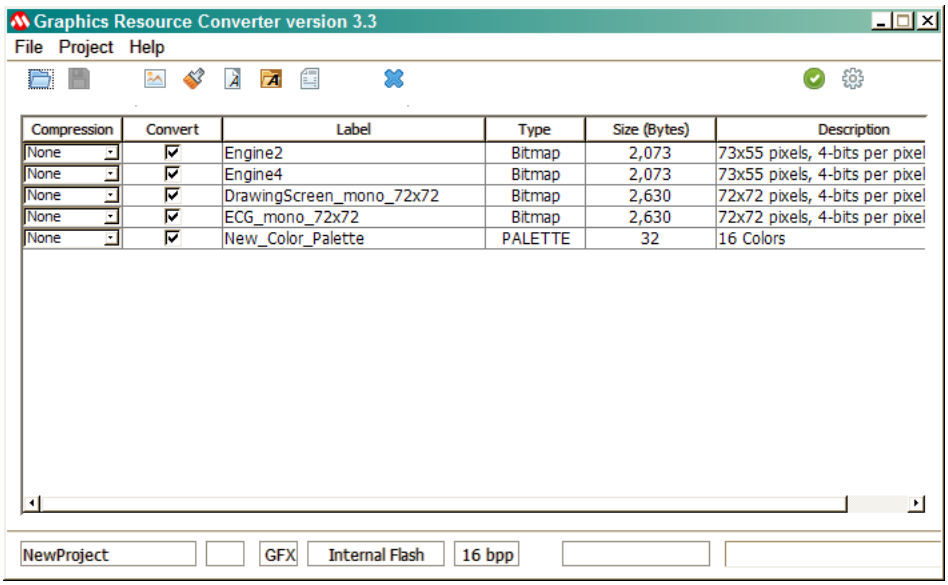




2. A Open dialog will appear and, by default, filters the project file (*.grp filter or *.xml filter).Type the project file name or browse for the project file that will be loaded and and press **Open** button.



3. After selecting the project, the resource table will be populated and the status will be updated.



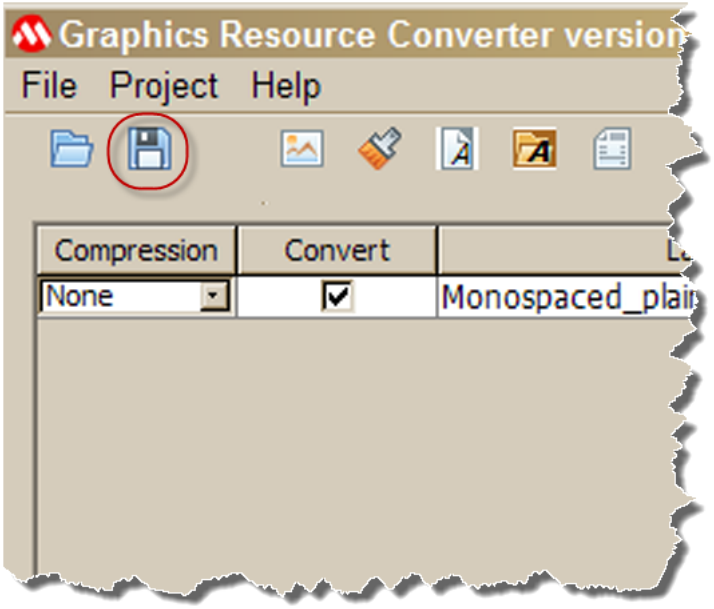
7.3.2 Saving

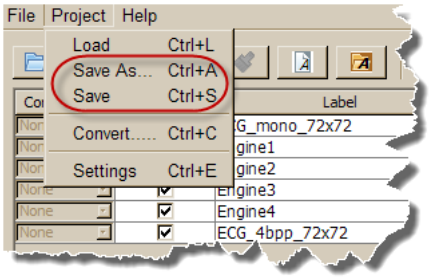
The utility allows users to save the loaded images, palettes and fonts into a project.

Saving a Project

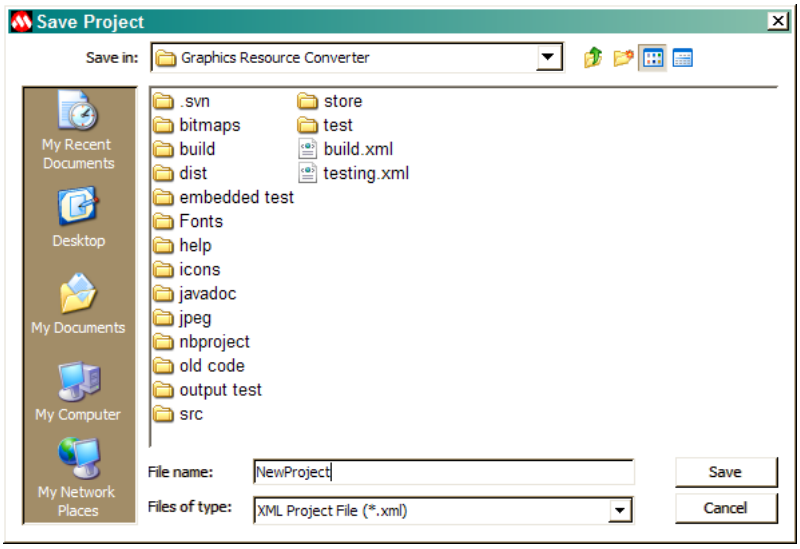
To save a project the following steps should be done:

1. Press **Save** button or Project Menu Item.

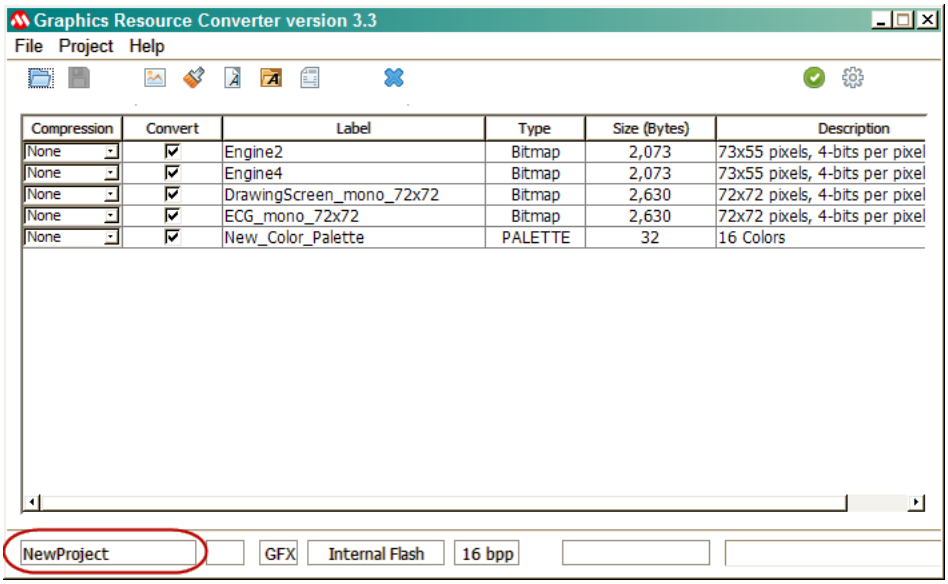




2. A Save As dialog will appear and, by default, filters the project file (*.xml filter).Type the new project file name or browse for the project file that will be overwritten and and press **Save** button.



3. The saved project will create a *.xml file and the project name will be placed in the status bar.

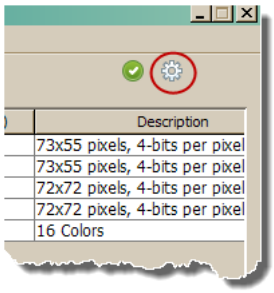


7.4 Settings

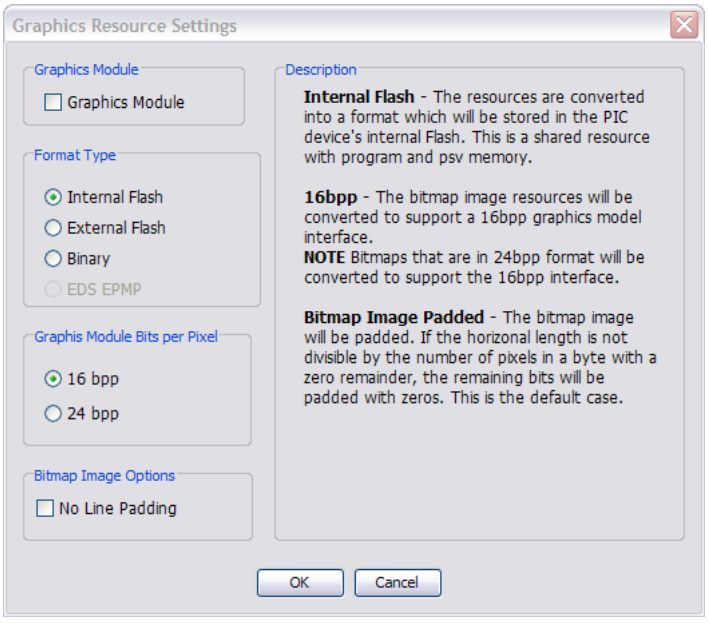
The Graphics Resource Converter can be configured for difference types of converting mediums along with different device builds.

To change the Graphics Resource Converter's settings

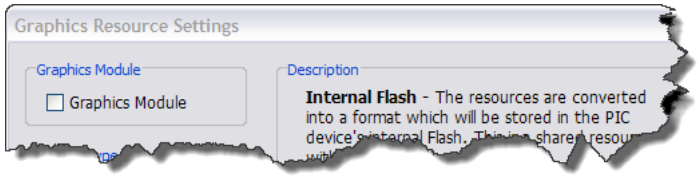
1. Press the **Settings** button.



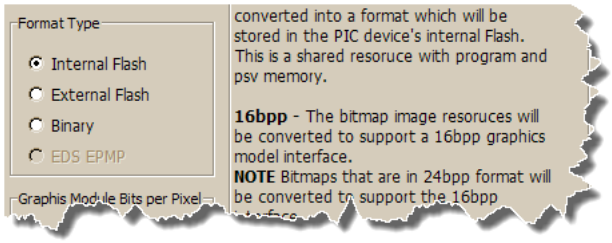
2. The Graphics Resource Setting dialog box will appear. The left side of the dialog has all of the configuration options, while the right side give a description of the configuration options that are currently selected.



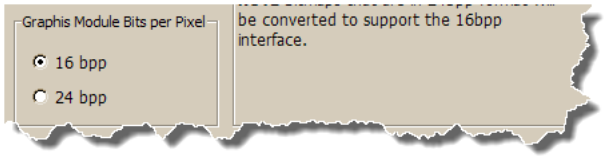
3. The Graphics Module panel contains the compiler build type
    1. Checking the **Graphics Module** will indicate that the device also has an internal graphics module. Please refer to the device's datasheet or Family Reference Manual to determine if it has an internal graphics module.



4. The Format Type panel contains the medium for the converted resources
    1. Internal Flash - The converted resources will be stored as C array structures. Proper device resources are needed to store these C array structures.
    2. External Flash - The converted resources will be stored as a Intel HEX file. This HEX file can be uploaded to an external memory source. The device will access this memory source to retrieve the resource data.
    3. Binary - The converted resources will be converted into a binary file.
    4. EDS EPMP - On devices with EDS memory space, the converted resources can be placed into this EDS space.



5. The Graphics Module Bits per Pixel option is for selecting the output of the converted image resources. Some graphics devices support different color depth. This option will allow the suer to select the color depth of 16bpp or 24bpp. It is import to note that this setting does not determine the color depth of the input resources. The application will down covert if needed. For example, a bitmap image that is 24bpp will be converted as a 16bpp if the Graphics Module Bits per Pixel setting is 16bpp.



6. The "No Line Padding" check box is use to select is there is not any padding for horizontal lines in bitmap images. This is useful when graphics controls that have a windowing option that auto increments.

7. After selecting the desired configuration settings, select **OK** and the status bar will populated with the current configuration settings. These configuration settings are saved with the project.

7.5 Converting Resources

When converting resources, the GRC has four output options, internal flash, external flash, binary and EDS PMP. These option provide a means to store the resources based on the application's resource requirements.

Internal Flash (see page 42)

The resource data will be stored as an array of data in a C source file. The data will be part of the flash or constant data. The size of the resources will impact the program data of the application as they are sharing the same resource space. Access to the data is as fast as the data bus.

External Flash (see page 45)

The resource data will be stored as part of an Intel HEX file. This file can be be uploaded to external memory device, like a SPI flash. The size of the resources is depended on the size of the external memory device. Access to the data is determined by the method use to retrieve it, topical slower than an internal data fetch. The application uses the generated reference output source file to obtain the location of a specific resource in the external memory.

Binary (see page 47)

The resource data will be stored as raw binary data. The binary file can be used by other applications or stacks to store the data. The application uses the generated reference output source file to obtain the offset of the specific resource in the extern memory. It is recommended if the intension of conversion is for external memory device, then use the external flash conversion setting.

EDS PMP (see page 50)

The resource data will be stored as part of an Intel HEX file. The file is to be uploaded to an external memory device that will use PMP to access it. This option is available only for devices with EDS memory space and a PMP module which is mapped to it. Access to the data is determined by the method use to retrieve it, topical slower than an internal data fetch. The application uses the generated reference output source file to obtain the location of a specific resource in the external memory and the EDS memory space.

7.5.1 Internal Flash

There is a limitation on the memory used when generating font images in C files (to be stored in internal flash for 16-bit PIC microcontroller devices). The font images are placed in the const section or program memory. The const section has a maximum size of 32 Kbytes. Therefore, the maximum size that the font image can have is 32 Kbyte assuming that no other data will reside in the const section or the font resources must be placed in program memory. When stored in external memory, the limitation will be the external memory size. The reason for storing fonts in the const section in internal flash is performance. It is faster to retrieve and display characters in the screen when placed in the const section.

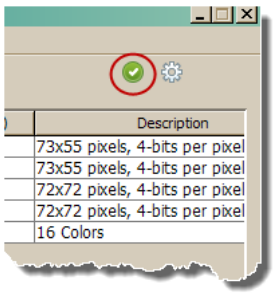
For PIC32, there is no limitation. As long as the internal flash has space you can pack in more font images.

Conversion of C file containing arrays to be located in internal flash memory is similar to the conversion of the Hex file.

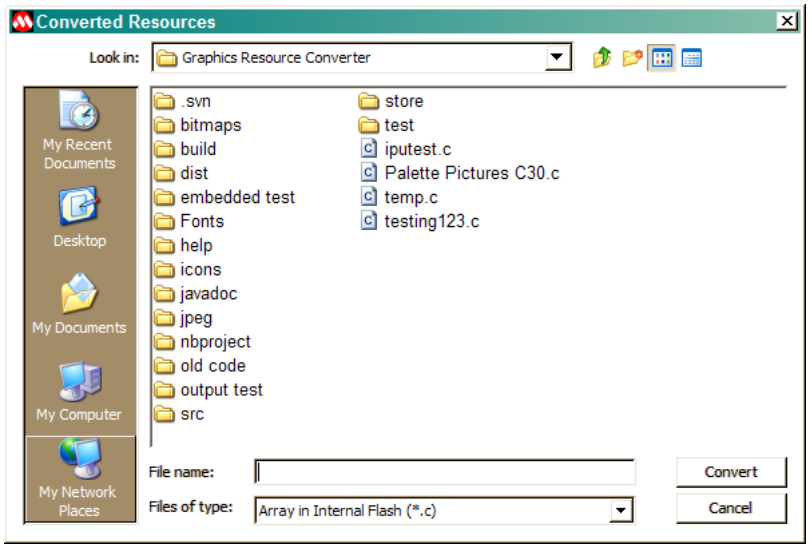
This conversion type should be used to store fonts and bitmaps into internal flash memory.

The following steps must be performed:

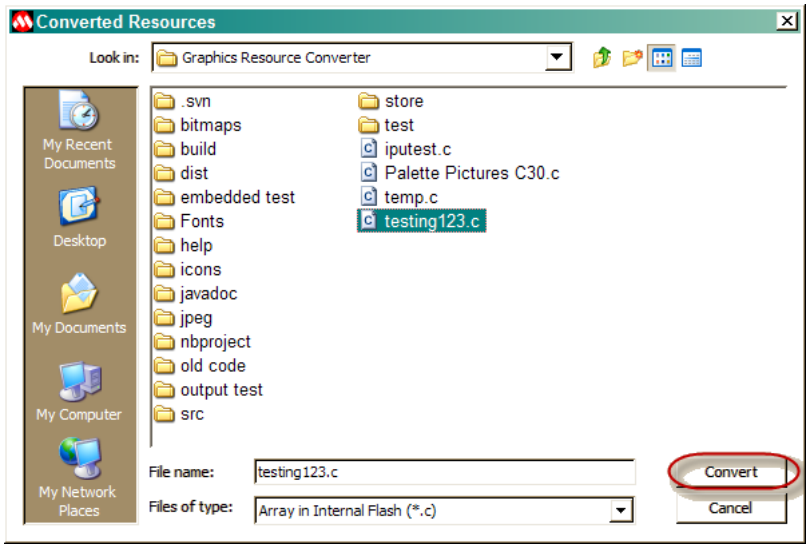
1. Press **Convert** button.



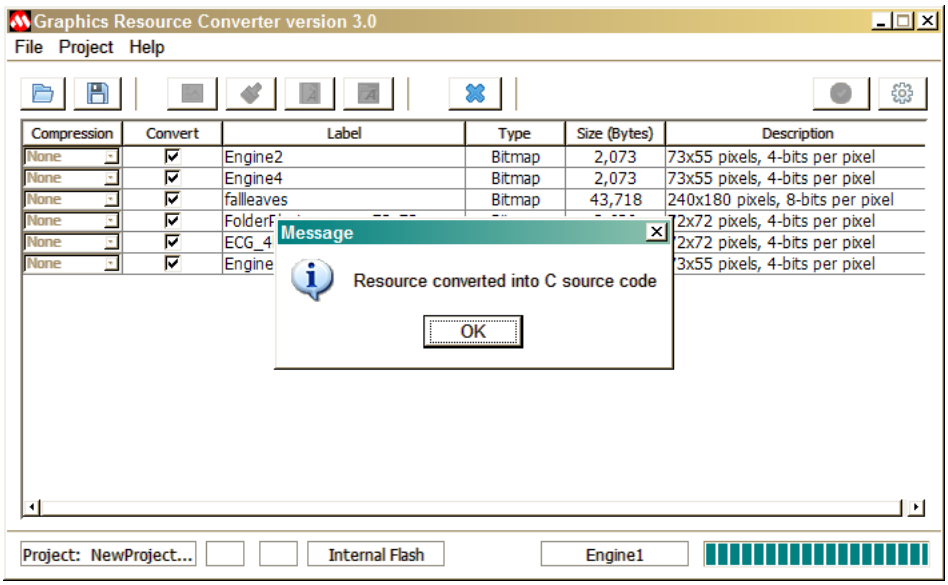
2. “Converted Resources” dialog will appear.



3. Press **Convert** button.



4. When the resources have been converted, a dialog will appear indicating such.



7.5.1.1 Reference Output

The follow structures are used by the GRC for images and fonts stored internally.

Internal Header Structure (Non-Compressed Resources)

Name	Bits	Description
Resource Description	16	See Resource Description Table (see page 52)
Address	32	The address in the internal memory where the resource is located

```
const IMAGE_FLASH DateAndTime_4bpp_72x72 =
{
    (FLASH | IMAGE_MBITMAP | COMP_NONE),
    (FLASH_BYTE *)__DateAndTime_4bpp_72x72
};
```

Internal Header Structure (Non-Compressed or Compressed Resources)

Name	Bits	Description
Resource Description	16	See Resource Description Table
ID	16	User defined ID
Address	32	The address in the internal memory where the resource is located
Width	16	The width of the image
Height	16	The height of the image
Parameter1	32	Parameter used for the resource. For example, a compressed image will have the compressed size, in bytes.
Parameter2	32	Parameter used for the resource. For example, a compressed image will have the uncompressed size, in bytes.
Color Depth	16	The color depth of an image

```
const GFX_IMAGE_HEADER DrawingScreen_4bpp_72x72 =
{
    (FLASH | IMAGE_MBITMAP | COMP_IPU),
    0,
```

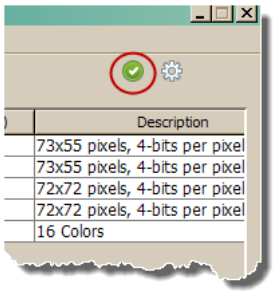
```
    {    .progByteAddress = (FLASH_BYTE *) __DrawingScreen_4bpp_72x72    },
    72,
    72,
    931,
    2630,
    4
};
```

7.5.2 External Flash

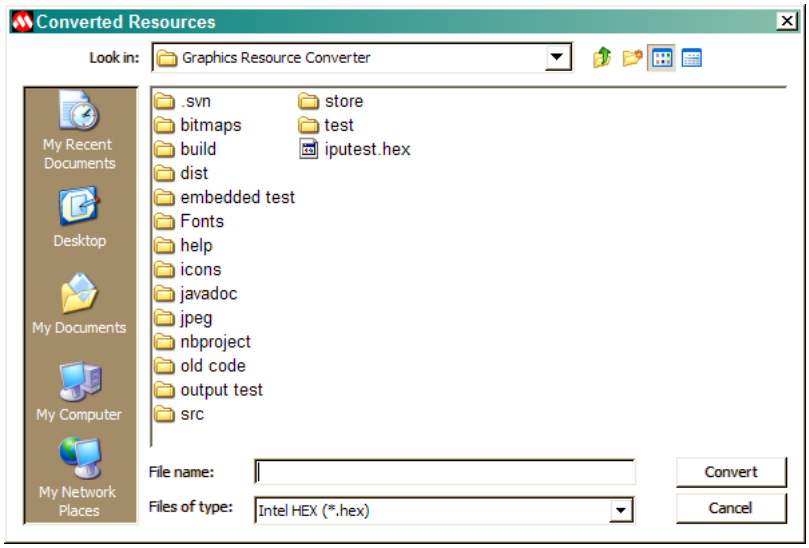
This output format should be selected to store bitmaps and fonts in external memory. Address of each object will be aligned by 4 bytes border. In addition to Intel hex file the utility will generate C file containing structures FONT_EXTERNAL and IMAGE_EXTERNAL. These structures must be used in application to access converted pictures and fonts in external memory. Name of the reference C file will be the same as name of the hex file with ".c" appended. This file will be compiled with the application to enable an easy reference to the strings that will be used in the application. Please refer to Font Reference File Output (see page 19) description.

To create Intel Hex and reference C files following steps must be performed:

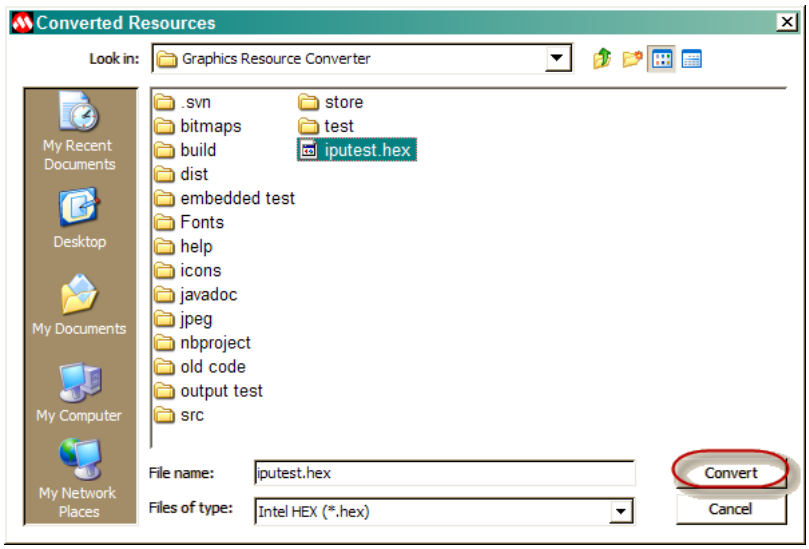
1. Press **Convert** button.



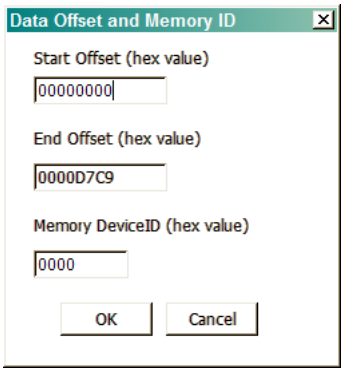
2. "Convert Resources" dialog will appear.



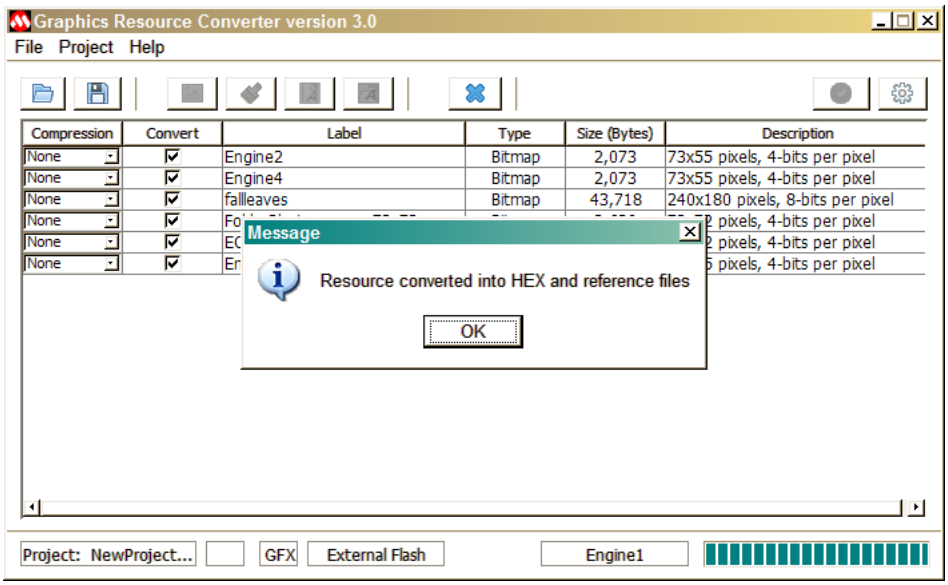
3. Press **Convert** button.



4. “Data offset and Memory ID” dialog will appear.



7. Enter start address if the converting data must have special location in the external memory.
8. Enter memory ID. Memory ID is a unique number assigned by application for the memory chip where data will be stored. This number allows distinguishing memory chip if the application has several of them.
9. Press **OK**. Conversion Complete message will appear when files are created successfully.



7.5.2.1 Reference Output

The follow structures are used by the GRC for images and fonts stored externally.

External Header Structure (Non-Compressed Resources)

Name	Bits	Description
Resource Description	16	See Resource Description Table (see page 52)
ID	16	User defined ID
Address	32	The address in the external memory where the resource is located

```
const IMAGE_EXTERNAL DateAndTime_4bpp_72x72 =
{
    (EXTERNAL | IMAGE_MBITMAP | COMP_NONE),
    0x0000,
    0x00000000
};
```

External Header Structure (Non-Compressed or Compressed Resources)

Name	Bits	Description
Resource Description	16	See Resource Description Table
ID	16	User defined ID
Address	32	The address in the external memory where the resource is located
Width	16	The width of the image
Height	16	The height of the image
Parameter1	32	Parameter used for the resource. For example, a compressed image will have the compressed size, in bytes.
Parameter2	32	Parameter used for the resource. For example, a compressed image will have the uncompressed size, in bytes.
Color Depth	16	The color depth of an image

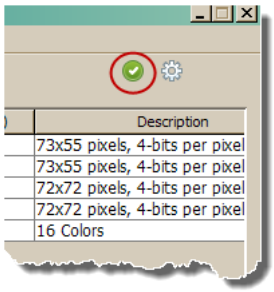
```
const GFX_IMAGE_HEADER DrawingScreen_4bpp_72x72 =
{
    (EXTERNAL | IMAGE_MBITMAP | COMP_IPU),
    0,
    {    .extAddress = (DWORD) 0x00000A50    },
    72,
    72,
    931,
    2632,
    4
};
```

7.5.3 Binary

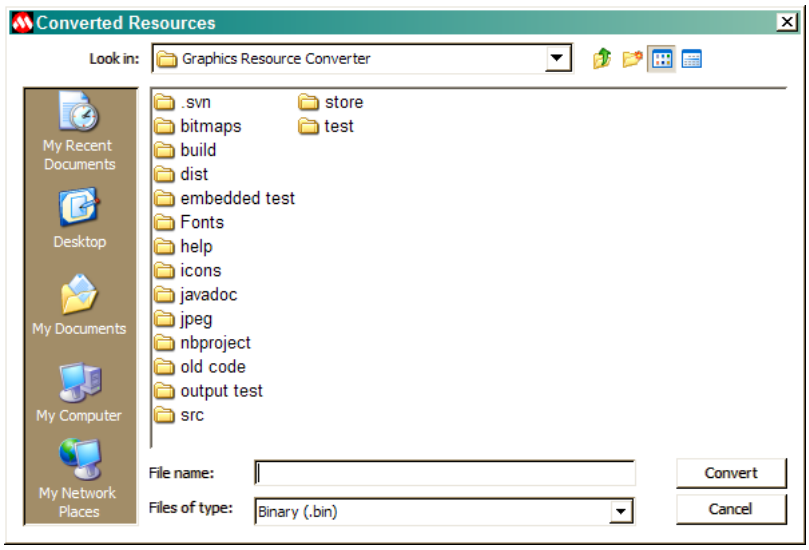
This output format allows creating binary files for resources in the conversion list. The output will be a binary file containing all the resource information, source reference and header file. The source reference file can be used to access resources (See Binary Reference Output section (see page 49)).

The following steps should be done to convert a GRC project to a binary output:

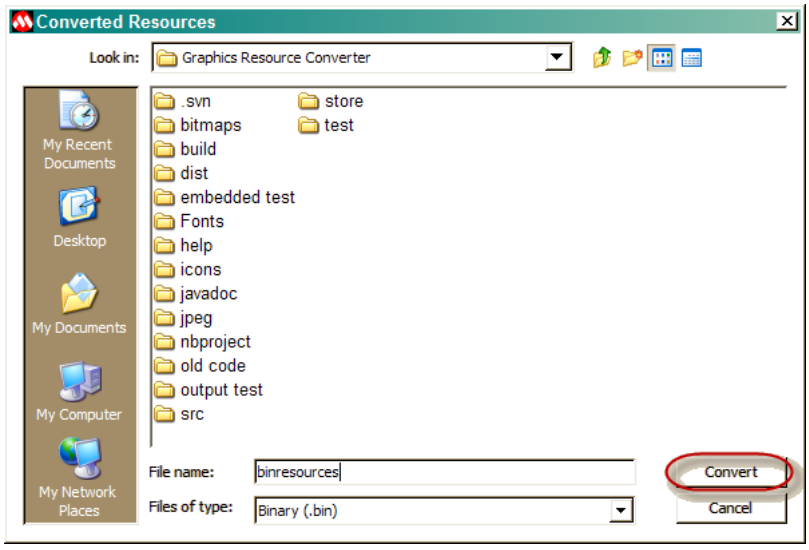
1. Press **Convert** button.



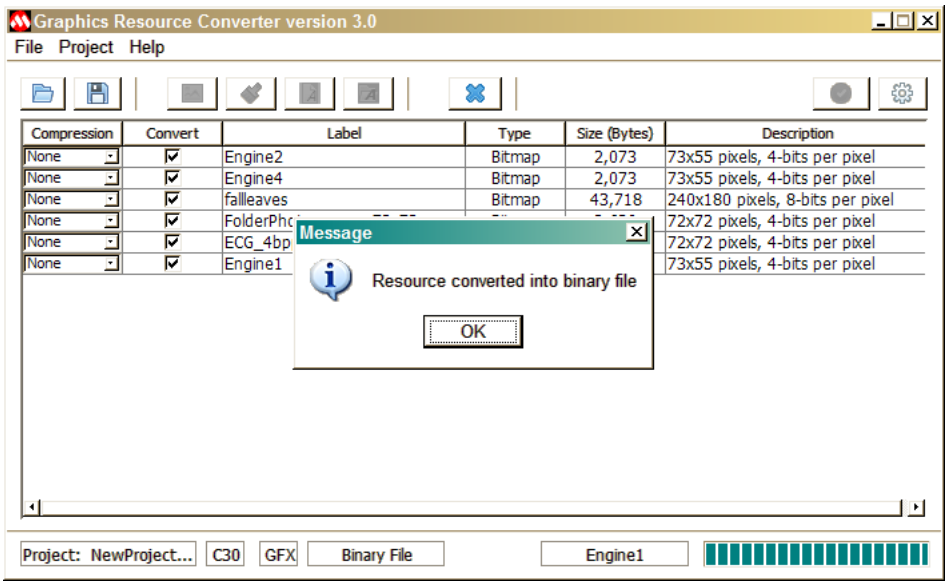
2. “Converted Resources” dialog will appear.



3. Press **Convert** button.



4. When the resources have been converted, a dialog will appear indicating such.



7.5.3.1 Reference Output

The follow structures are used by the GRC for images and fonts stored in a binary file.

External Header Structure (Non-Compressed Resources)

Name	Bits	Description
Resource Description	16	SeeResource Description Table (see page 52)
ID	16	User defined ID
Offset	32	Location from the start of the file.

```
const IMAGE_EXTERNAL DateAndTime_4bpp_72x72 =
{
    (EXTERNAL | IMAGE_MBITMAP | COMP_NONE),
    0x0000,
    0x00000000
};
```

External Header Structure (Non-Compressed or Compressed Resources)

Name	Bits	Description
Resource Description	16	See Resource Description Table
ID	16	User defined ID
Offset	32	Location from the start of the file.
Width	16	The width of the image
Height	16	The height of the image
Parameter1	32	Parameter used for the resource. For example, a compressed image will have the compressed size, in bytes.
Parameter2	32	Parameter used for the resource. For example, a compressed image will have the uncompressed size, in bytes.
Color Depth	16	The color depth of an image

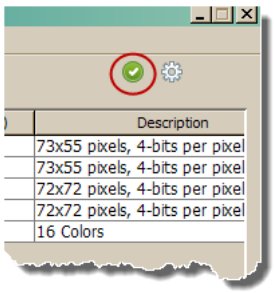
```
const GFX_IMAGE_HEADER DrawingScreen_4bpp_72x72 =
{
    (EXTERNAL | IMAGE_MBITMAP | COMP_IPU),
    0,
    {    .extAddress = (DWORD) 0x00000A48    },
    72,
    72,
    931,
    2632,
    4
};
```

7.5.4 EDS PMP

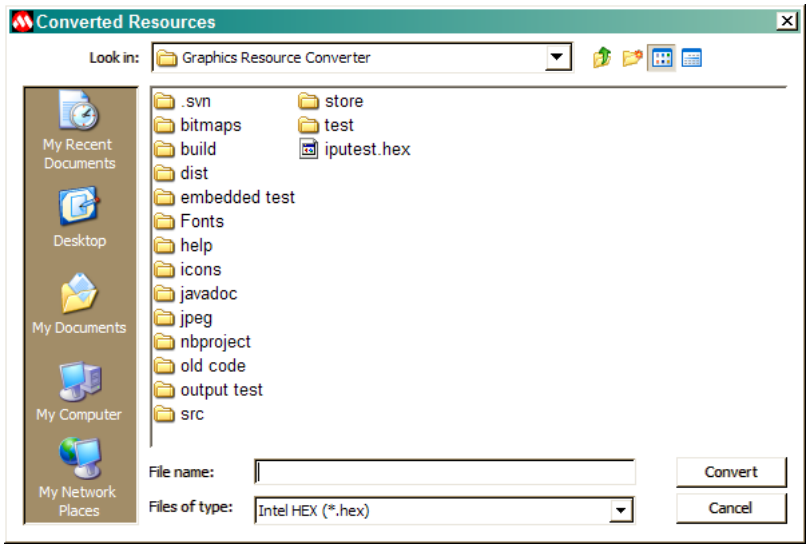
This output format should be selected to store bitmaps and fonts in external memory. Address of each object will be aligned by 4 bytes border. In addition to Intel hex file the utility will generate C file containing structures FONT_EXTERNAL and GFX_IMAGE_HEADER. These structures must be used in application to access converted pictures and fonts in external memory. Name of the reference C file will be the same as name of the hex file with ".c" appended. This file will be compiled with the application to enable an easy reference to the strings that will be used in the application. Please refer to EDS PMP Reference Ouptut (see page 52) description. The device that the resources are being converted to must have EDS space access through EPMP. Please check the device datasheet for this information.

To create Intel Hex and reference C files following steps must be performed:

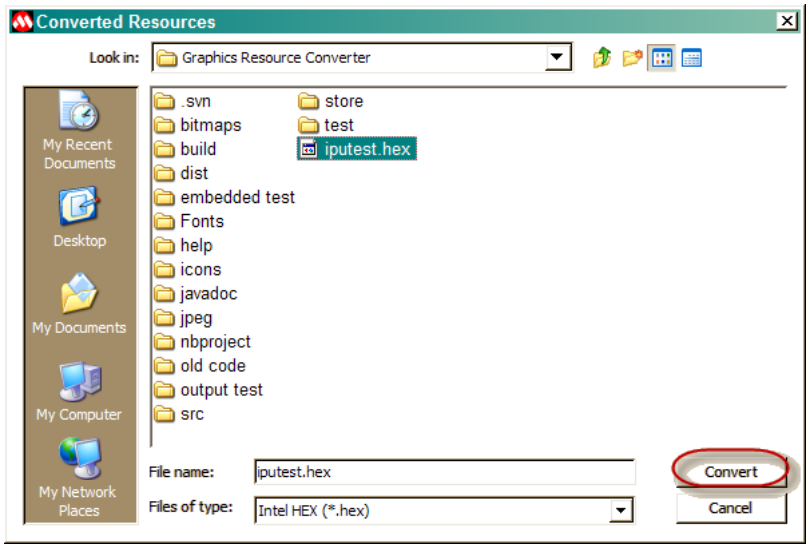
1. Press **Convert** button.



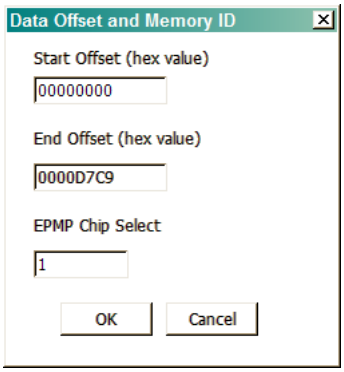
2. "Convert Resources" dialog will appear.



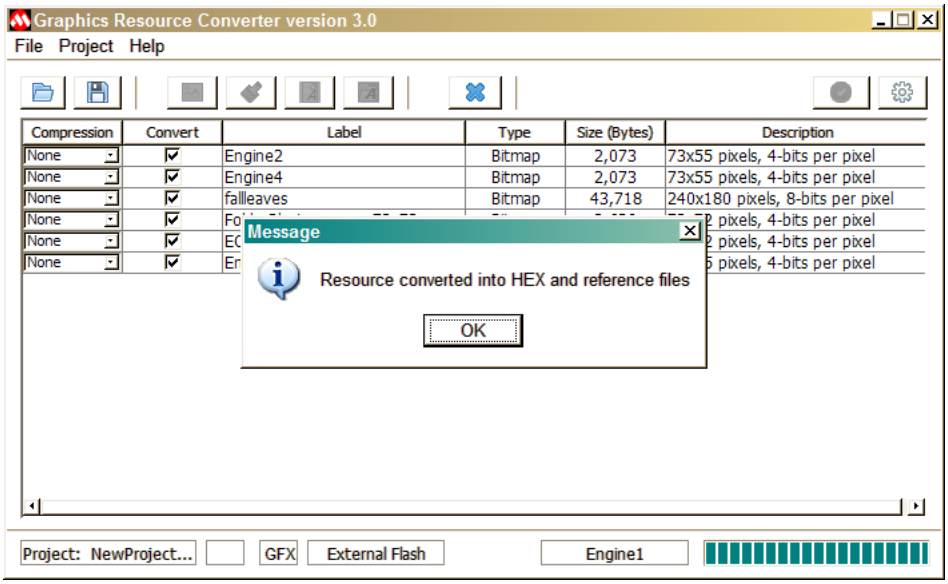
3. Press **Convert** button.



4. “Data offset and Memory ID” dialog will appear.



7. Enter start address if the converting data must have special location in the external memory.
8. Enter EPMP Chip Select. The EPMP Chip Select is the value of the chip select being used. Zero is not a valid number for the chip select.
9. Press **OK**. Conversion Complete message will appear when files are created successfully.



7.5.4.1 Reference Output

The follow structures are used by the GRC for images and fonts stored in EDS space.

External Header Structure (Non-Compressed or Compressed Resources)

Name	Bits	Description
Resource Description	16	SeeResource Description Table (see page 52)
ID	16	User defined ID
Address	32	The address in the external memory where the resource is located
Width	16	The width of the image
Height	16	The height of the image
Parameter1	32	Parameter used for the resource. For example, a compressed image will have the compressed size, in bytes.
Parameter2	32	Parameter used for the resource. For example, a compressed image will have the uncompressed size, in bytes.
Color Depth	16	The color depth of an image

```
const GFX_IMAGE_HEADER DrawingScreen_4bpp_72x72 =
{
    (EDS_EPMP | IMAGE_MBITMAP | COMP_IPU),
    1,
    {    .edsAddress = (__eds__ char *) (0x00000A50 + GFX_EPMP_CS1_BASE_ADDRESS)   },
    72,
    72,
    931,
    2632,
    4
};
```

7.5.5 Resource Description Table

Every resource table contains a resource description entry. This entry describes the location, type and compression used in a resource.

Resource Description Table

Name	Bits	Description
Location	0-7	The location of the resource. FLASH - resource is located in internal flash EXTERNAL - resource is located in an external memory device RAM - resource is located in internal RAM EDS_EPMP - resource is located in an external memory device which is accessed used eds memory space and PMP.
Type	8-11	The type of resource. IMAGE_MBITMAP - The resource is a Microchip converted bitmap IMAGE_JPEG - The resources is a JPEG image

Compression	12-15	The compression used on the resource. `COMP_NONE` - no compression is used on the resource `COMP_RLE` - Run-Length Encoding (RLE) compression is used. This is only valid on bitmap images that are 8 or 4 bpp. `COMP_IPU` - The Deflate algorithm is used to compress the image.

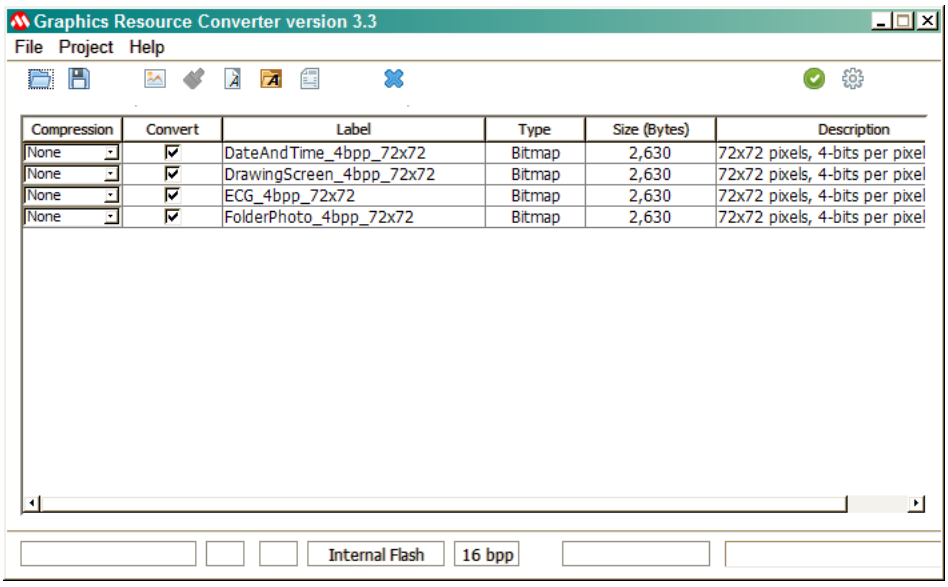
7.6 Generating a Palette from Bitmap Images

The converter can generate a palette based on bitmap images that are loaded into the resource table. If there are any JPEG images, the palette obtain will not be available. The palette that is generated will be up to 256 colors and the bitmap images will be reformatted to use the generated palette.

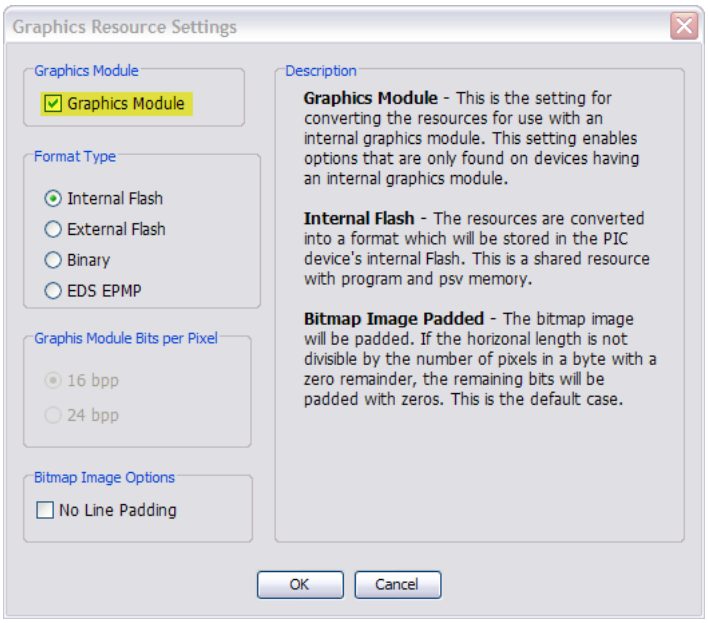
Generating a Palette

To generate a palette follow these steps:

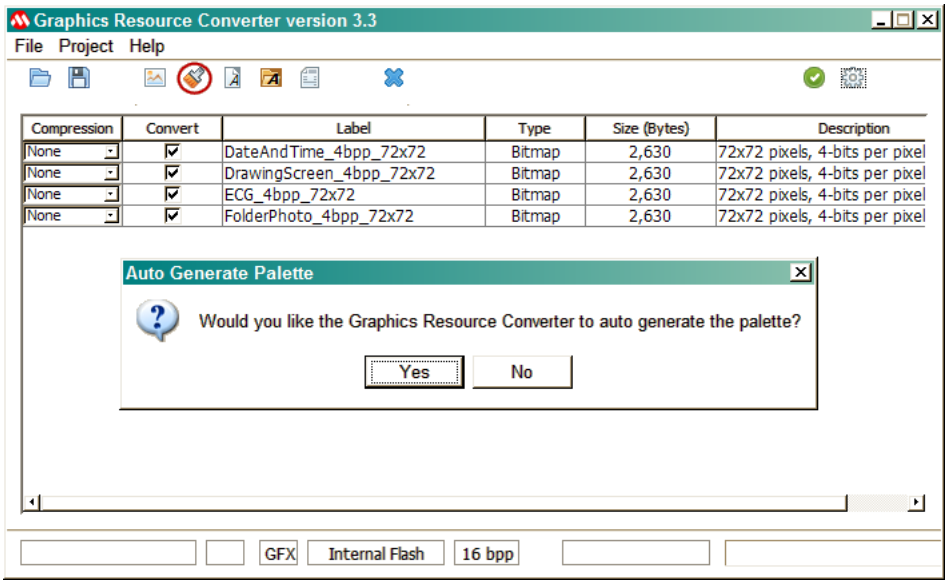
1. Add all of the bitmap images to the converter for the project.



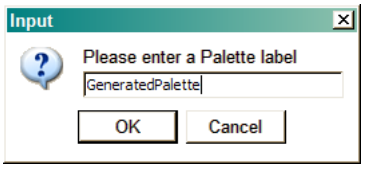
2. Under the settings dialog box, make sure that the graphics module option is selected. Palettes will not be generated for projects that do not have a graphics module.



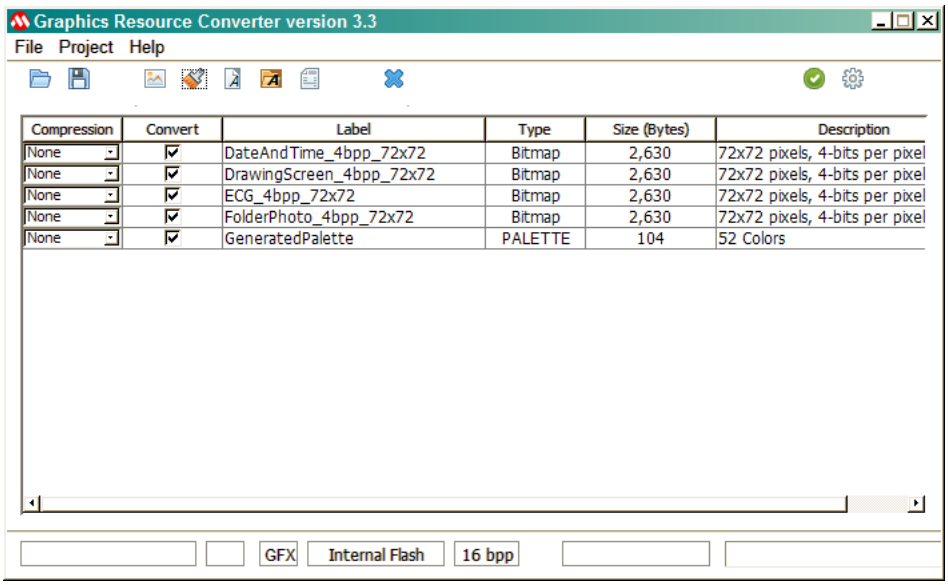
3. When prompted, choose **YES** to allow the converter to auto generate the palette.



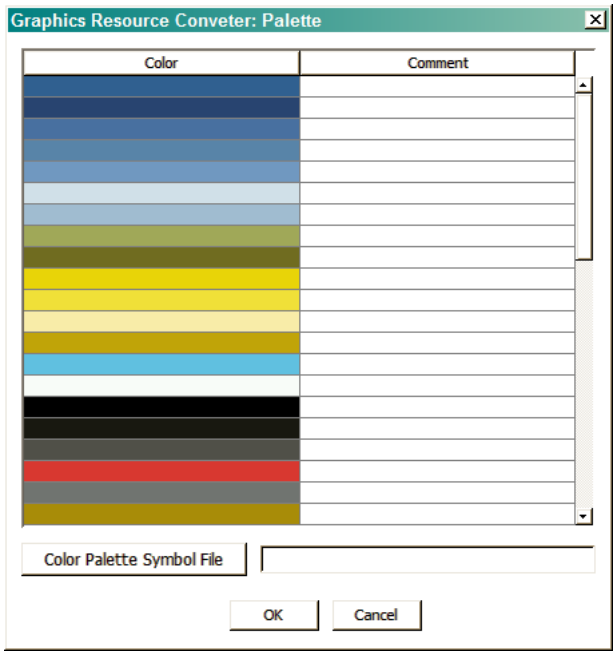
4. Enter the palette label that you wish to use.



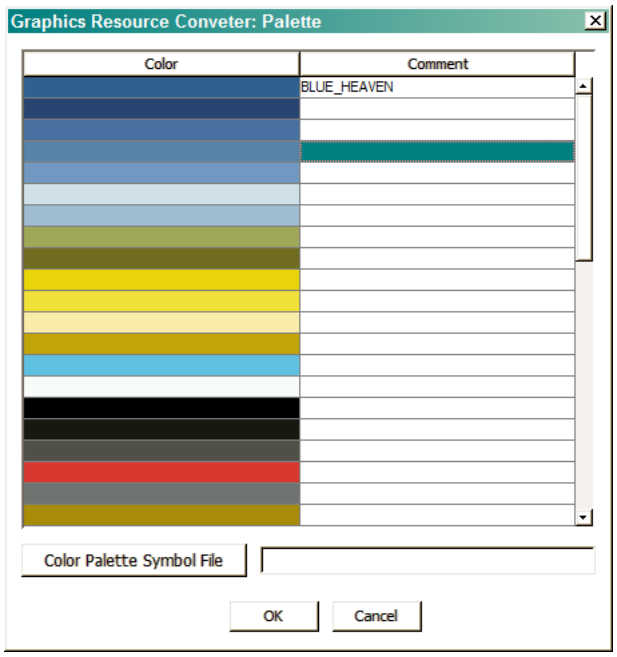
5. You will see that a palette has been generated from the individual bitmap images.



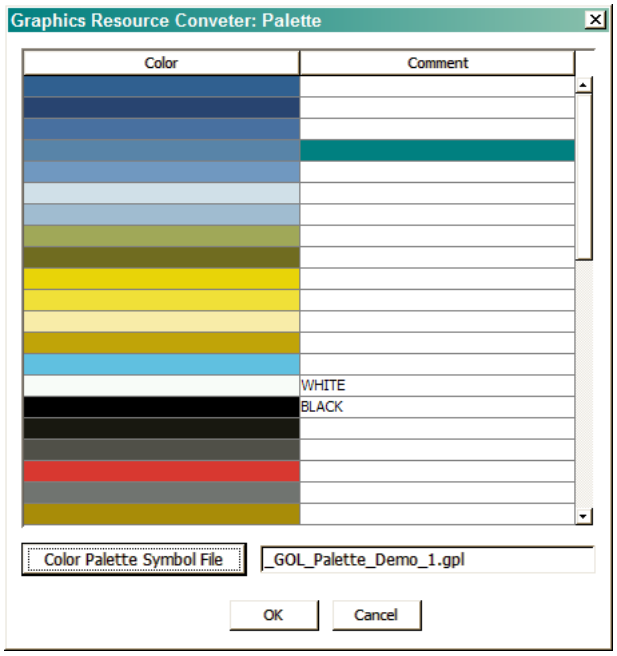
6. By double clocking on the palette resource in the resource table, you will bring up the color chart.



7. You can manually add in comments that, when converted will become part of a palette color defines header file.
    1. The application can include this file to reference the color palette index via a symbol



8. Selecting the Color Palette Symbol File button will load a GIMP palette and use the comments in the file.
    1. The generated color palette will match the colors in the GIMP palette file to place the comments



9. Convert the project into the medium of choice.

7.7 Using the Command Line Interface

Starting with Graphics Resource Converter version 3.3, a command line interface is supported. This interface allows for resource conversion to be done without launching the GUI. The JAVA runtime engine (JRE) is still requirement to run the command line interface.

7.7.1 Command Line Interface

The Graphics Resource Converter, GRC, can be run through a command line interface. Passing arguments will determine if the GUI is launched or the command line interface is used.

To run the GRC GUI from the command line:

```
>java –jar “<Microchip Applications Library directory>/Microchip/Graphics/bin/grc/grc.jar”
```

To run the GRC without launching the GUI:

```
>java –jar “<Microchip Applications Library directory>/Microchip/Graphics/bin/grc/grc.jar”
<options>
```

where <options> are the GRC command line options.

Here is an example of some command line options:

```
java -jar "<my grc path>/grc.jar" -T I -F C -I "images/DateAndTime_4bpp_72x72.bmp" -O
"compression/commandline_test_carray.c" -X DandT_Uncompressed_4bpp ; -T I -F C -I
"images/DateAndTime_4bpp_72x72.bmp" -O "compression/commandline_test_carray.c" -X
DandT_Compressed_RLE_4bpp -P RLE ; -T I -F C -I "images/DateAndTime_8bpp_72x72.bmp" -O
"compression/commandline_test_carray.c" -X DandT_Uncompressed_8bpp ; -T I -F C -I
"images/DateAndTime_8bpp_72x72.bmp" -O "compression/commandline_test_carray.c" -X
DandT_Compressed_RLE_8bpp -P RLE ; -T I -F C -I "images/sunset1.bmp" -O
"compression/commandline_test_carray.c" -X sunset_Uncompressed ; -T I -F C -I
"images/sunset1.bmp" -O "compression/commandline_test_carray.c" -X sunset_Compressed_IPU -P
IPU
```

7.7.2 Command Line Options

All command line options and associated values must be separated by a space. If the value of the argument contains a space, it must be surrounded by quotes.

For example:

```
Resource Type: Image
-T I -- This command line switch is –T and the value is I.
Output File
-O “output file.c” – This command line switch is –O and the value is “output file.c.” Since
the value has a space, it must be surrounded by quotes
```

Resource Type (-T)

I – An image, either bitmap or JPEG.

F – A font, either an installed or font file.

Resource Sub-type (-U)

I – installed font

T – TTF font file

R – raster font file

Output Format (-F)

C – C array

H – Hex

B – Binary

Input File (-I)

The value will be the file name and path. The path can be relative or absolute. It is recommended to surround the file and path with quotes.

Output File (-O)

The value will be the file name and path. The path can be relative or absolute. It is recommended to surround the file and path with quotes.

Font Start Character (-S)

The value is the starting character of the font.

Font End Character (-E)

The value is the ending character of the font.

Font Style Italic (-A)

Font Style Bold (-D)

Font Extended Glyph (-FEG)

Use the extended Glyph entry for the characters.

Font Anti-Alaising (-FAA)

Use anti-aliasing when creating the font characters.

Font Name (-N)

The value is the font name. For example, “Times New Roman” is the name of the font used to write this document.

Font Size (-Z)

The value is the font size.

Font Filter File (-R)

The value will be the file name and path. The path can be relative or absolute. It is recommended to surround the file and path with quotes.

Compression (-P)

IPU – Deflate compression, only for DA210 applications

RLE – Run length encoding

Label (-X)

The value should start with a letter and have not spaces.

Project File (-XML)

The GRC project file to convert.

Hex Address Start (-HA)

The starting address of the generated Hex file.

EDS Chip Select (-CS)

The chip select of the EDS PMP.

Version (-VERSION)

Replies with the version of the GRC in use. This should be the only argument passed.

Break (;)

Index